Schema Refinement and Normal Forms

courtesy of Joe Hellerstein for some slides

Jianlin Feng
School of Software
SUN YAT-SEN UNIVERSITY

Review: Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level description (often done with ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - consistency, normalization
- Physical Design - indexes, disk layout
- Security Design - who accesses what

The Evils of Redundancy 冗余

- *Redundancy* is at the root of several problems associated with relational schemas:
  - *redundant storage, insert/delete/update anomalies*
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition*
  - replacing ABCD with, say, AB and BCD, or ACD and ABD.
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Functional Dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation schema R if, for every allowable instance *r* of R:

$$t1 \in r,\ t2 \in r,\ \pi_X(t1) = \pi_X(t2)$$

$$\text{implies}\quad \pi_Y(t1) = \pi_Y(t2)$$

(where t1 and t2 are tuples;X and Y are sets of attributes)

- In other words: $X \rightarrow Y$ means

  Given any two tuples in *r*, if the X values are the same, then the Y values must also be the same. (but not vice versa)

- Can read "$\rightarrow$" as "determines"

Functional Dependencies (Contd.)

- An FD is a statement about *all* allowable relations.
  - Must be identified based on semantics of application.
  - Given some instance *r1* of R, we can check if *r1* violates some FD *f*, but we cannot determine if *f* holds over R.
- Question: How related to keys?
  - if “K $\rightarrow$ all attributes of R” then K is a *superkey* for R (does not require K to be *minimal*.)
- FDs are a generalization of keys.

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:

Hourly_Emps (***<u>ssn</u>*, *name*, *lot*, *rating*, *wage_per_hr***, ***hrs_per_wk***)

- We sometimes denote a relation schema by listing the attributes: e.g., SNLRWH
- Sometimes, we refer to the set of *all attributes* of a relation by using the relation name. e.g., "Hourly_Emps" for SNLRWH

What are some FDs on Hourly_Emps?

ssn is the key: $S \rightarrow SNLRWH$

rating determines *wage_per_hr*: $R \rightarrow W$

lot determines *lot*: $L \rightarrow L$ ("trivial" dependnency)

Problems Due to R $\rightarrow$ W

S	N	L	R	W	H
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

Hourly_Emps

- *Update anomaly*: Can we modify W in only the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his or her rating? (or we get it wrong?)
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Detecting Redundancy

S	N	L	R	W	H
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

Hourly_Emps

Q: Why was R $\rightarrow$ W problematic, but S $\rightarrow$ W not?

Decomposing a Relation 分解关系

- Redundancy can be removed by "chopping" the relation into pieces. 消除冗余
- FD's are used to drive this process.

R $\rightarrow$ W is causing the problems, so decompose SNLRWH into what relations?

S	N	L	R	H
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

Hourly_Emps2

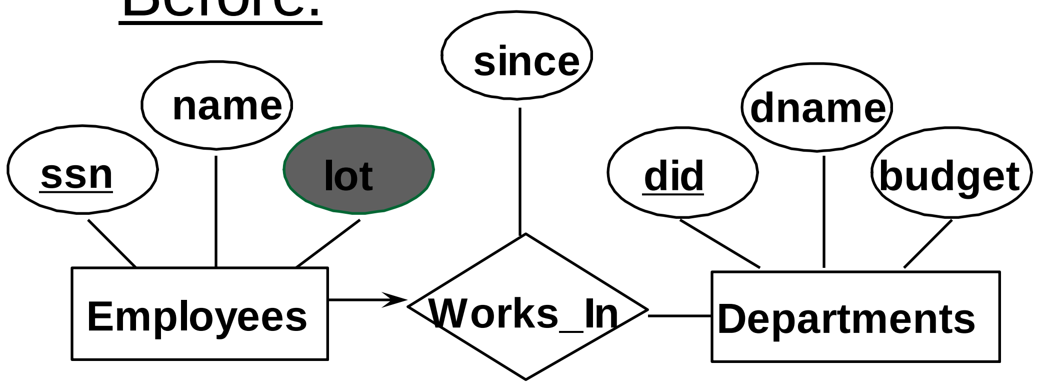
R	W
8	10
5	7

Wages

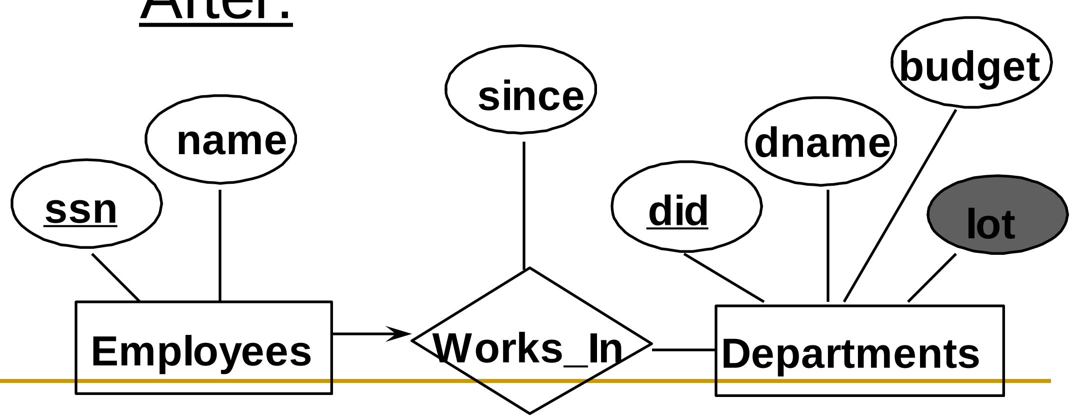
Refining an ER Diagram by FD: Attributes Can Easily Be Associated with the "Wrong" Entitity Set in ER Design.

- 1st diagram becomes:
  Workers(S,N,L,D,Si)
  Departments(D,M,B)
  - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot: $D \rightarrow L$
- Redundancy; fixed by decomposition:
  Workers2(S,N,D,Si)
  Dept_Lots(D,L)
  Departments(D,M,B)
- Can fine-tune this:
  Workers2(S,N,D,Si)
  Departments(D,M,B,L)

Before:



After:



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:

  title $\rightarrow$ *studio, star* implies *title* $\rightarrow$ *studio* and *title* $\rightarrow$ *star*

  title $\rightarrow$ *studio* and *title* $\rightarrow$ *star* implies *title* $\rightarrow$ *studio, star*

  title $\rightarrow$ *studio, studio* $\rightarrow$ *star* implies *title* $\rightarrow$ *star*

  But, *title, star* $\rightarrow$ *studio* does NOT necessarily imply that *title* $\rightarrow$ *studio* or that *star* $\rightarrow$ *studio*

- An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.
- F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")

Rules of Inference

- Armstrong's Axioms (X, Y, Z are <u>sets</u> of attributes):
  - *Reflexivity*: If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- These are *sound* and *complete* inference rules for FDs!
  - i.e., using AA you can compute all the FDs in F+ and only these FDs.
- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Example

- Contracts(*<u>cid</u>,sid,jid,did,pid,qty,value*), and:
  - C is the key: $C \to CSJDPQV$
  - P(roject) purchases a given part using a single contract: $JP \to C$
  - D(ept) purchases at most 1 part from a supplier: $SD \to P$

- Problem: Prove that SDJ is a key for Contracts
  - $JP \to C$, $C \to CSJDPQV$ imply $JP \to CSJDPQV$ (by transitivity) (shows that JP is a key)
  - $SD \to P$ implies $SDJ \to JP$ (by augmentation)
  - $SDJ \to JP$, $JP \to CSJDPQV$ imply $SDJ \to CSJDPQV$ (by transitivity) thus SDJ is a key.

Attribute Closure

- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F. An efficient check:
  - Compute *attribute closure* of X (denoted X^+) w.r.t. F.
    X^+ = Set of all attributes A such that $X \rightarrow A$ is in F^+
    - $X^+ := X$
    - Repeat until no change: if there is an FD $U \rightarrow V$ in F such that U is in X^+, then add V to X^+
  - Check if Y is in X^+
- The approach can also be used to find the keys of a relation.
  - If all attributes of R are in the closure of X, then X is a superkey for R.

Attribute Closure (example)

- $R = \{A, B, C, D, E\}$
- $F = \{ B \to CD, D \to E, B \to A, E \to C, AD \to B \}$
- Is $B \to E$ in F^+ ?

  $B^+ = B$

  $B^+ = BCD$

  $B^+ = BCDA$

  $B^+ = BCDAE$ … Yes!

  and B is a key for R too!
- Is D a key for R?

  $D^+ = D$

  $D^+ = DE$

  $D^+ = DEC$

  … Nope!

- **Is AD a key for R?**

  $AD^+ = AD$

  $AD^+ = ABD$ and B is a key, so Yes!
- **Is AD a *candidate* key for R?**

  $A^+ = A$

  A not a key, nor is D so Yes!
- **Is ADE a *candidate* key for R?**

  No! AD is a key, so ADE is a superkey, but not a candidate key.

Normal Forms

- How to do schema refinement?
  - We use normal forms as guidance.

- If a relation is in a normal form (BCNF, 3NF etc.):
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.

Normal Forms vs. Functional Dependencies

- Role of FDs in detecting redundancy:
  - Consider a relation *R* with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given $A \rightarrow B$: If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!
- The normal forms based on FDs are:
  - First Normal Form (1NF), 2NF, 3NF, and Boyce-Codd Normal Form (BCNF)
  - These forms have increasingly restrictive requirements:
    - $1^{NF} \supset$ 2NF (of historical interest) $\supset$ 3NF $\supset$ BCNF

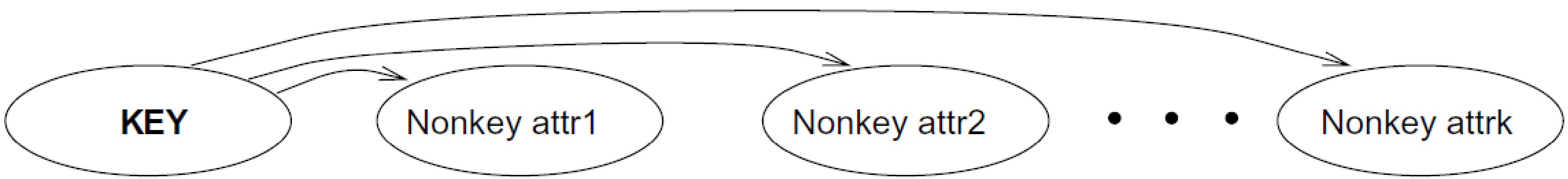
1st Normal Form

- 1st Normal Form – all attributes are atomic
  - Given a relation R in 1NF, for a tuple *t* of *R*, *t'*s every attribute can contain only atomic values,
  - i. e., attribute values are not lists or sets.

- We can imagine there exists FDs in the following manner:
  - Attribute *A* → *some unique value in A*'s *domain*.

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey for R.
- Intuitively, R is in BCNF if the only non-trivial FDs over R are *key constraints*.



Boyce-Codd Normal Form (Contd.)

- If R in BCNF, then every field of every tuple records information that cannot be inferred using FDs alone.
  - Say we know FD X $\rightarrow$ A holds for this example relation:

X	Y	A
x	y1	a
x	y2	?

- Can you guess the value of the missing attribute?
- Yes, so relation is not in BCNF

Decomposition of a Relation Scheme

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.

- Suppose that relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R,
  - and every attribute of R appears as an attribute of at least one of the new relations.

Example (same as before)

S	N	L	R	W	H
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

Hourly_Emps

- SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
- Q: Is this relation in BCNF?

No, The second FD causes a violation;
W values repeatedly associated with R values.

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
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

Hourly_Emps2

R	W
8	10
5	7

Wages

- Q: Are both of these relations are now in BCNF?
- **Decompositions should be used only when needed.**
  - Q: potential problems of decomposition?

Problems with Decompositions

- There are three potential problems to consider:
  1) May be impossible to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require JOINs.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

***Tradeoff*:** **Must consider these issues vs. redundancy.**

Lossless Decomposition (example)

S	N	L	R	H
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

$\bowtie$

R	W
8	10
5	7

=

S	N	L	R	W	H
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

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$A \rightarrow B;\ C \rightarrow B$

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B
1	2
4	5
7	2

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Join Decompositions

无损分解

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:

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

- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem #1)*

Simple Test on Lossless Decomposition

- The decomposition of R into X and Y is lossless w. r. t. F *iff* the closure of F contains:

  $$X \cap Y \rightarrow X, \quad \textbf{or}$$

  $$X \cap Y \rightarrow Y$$

- In the example: decomposing ABC into AB and BC is lossy, because intersection (i.e., “B”) is not a key of either resulting relation.

- Useful result: If $W \rightarrow Z$ holds over R and $W \cap Z$ is empty, then decomposition of R into R-Z and WZ is loss-less.

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	C
1	3
4	6
7	8

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Dependency Preserving Decomposition

- Intuitively, a dependency preserving decomposition allows us to enforce all FDs by examining a single relation instance on each insertion or modification of a tuple.
  - *(Avoids Problem #2 on our list.)*

- *Projection of set of FDs F* :
  - If R is decomposed into X and Y,
  - the projection of F on X (denoted F_X) is the set of FDs $U \rightarrow V$ in F^+ such that all of the attributes U, V are in X. *(same holds for Y of course)*

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is ***dependency preserving*** if $(F_X \cup F_Y)^+ = F^+$.

- Important to consider F^+ in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?
    - note: F^+ contains $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$, so…

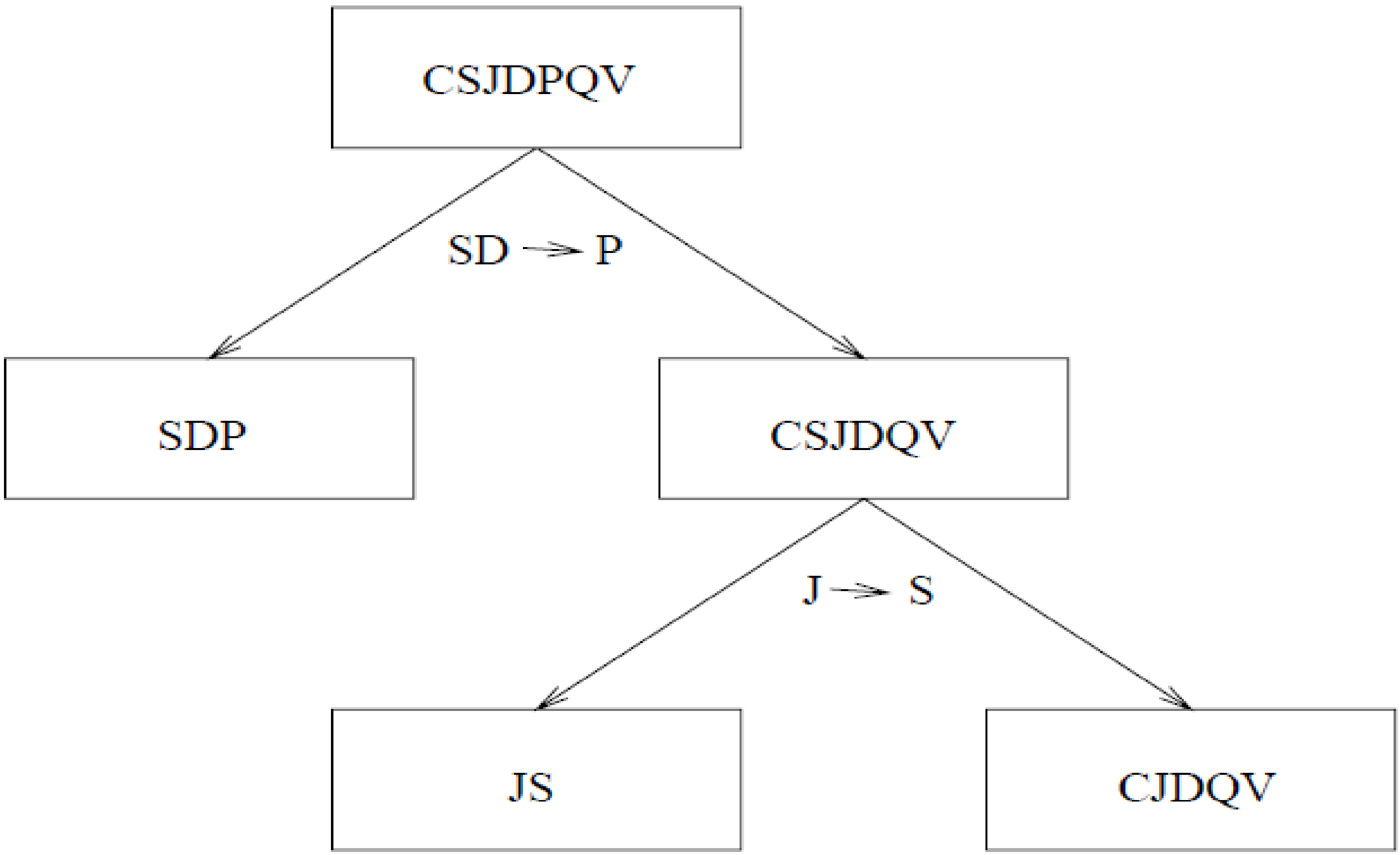
- F_{AB} contains $A \rightarrow B$ and $B \rightarrow A$; F_{BC} contains $B \rightarrow C$ and $C \rightarrow B$ .So, $(F_{AB} \cup F_{BC})^+$ contains $C \rightarrow A$

Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow Y$ violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).
  - Repeated application of this idea will give us a collection of relations that are in BCNF;
  - lossless join decomposition, and guaranteed to terminate.

  - e.g., CSJDPQV, key C, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - *{contractid, supplierid, projectid, deptid, partid, qty, value}*
  - To deal with $SD \rightarrow P$, decompose into SDP, CSJDQV.
  - To deal with $J \rightarrow S$, decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV

Decomposition of CSJDPQV into SDP, JS, and CJDQV



What if we do the decomposition by using the dependency J→S first?

BCNF and Dependency Preservation

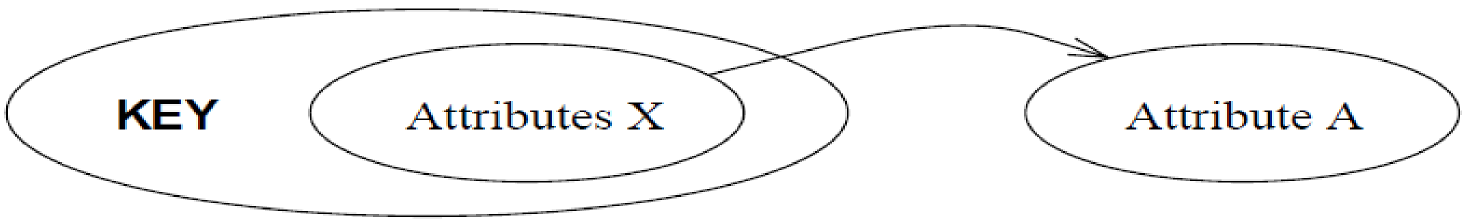
- In general, there may not be a dependency preserving decomposition into BCNF.
  - decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs $JP \rightarrow C$, $SD \rightarrow P$ and $J \rightarrow S$).
  - i.e., the dependency $JP \rightarrow C$ can not be enforced without a join. However, it is a lossless join decomposition.
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - but JPC tuples are stored only for checking the FD. (*Redundancy!*)

Third Normal Form (3NF)

- Relation R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when no "good" decomposition for BCNF, or for performance considerations.
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

3NF Violated by X→A: Case 1

- X is a proper subset of some key K
  - Such X→A is sometimes called a partial dependency.
    - We store (X, A) pairs redundantly.

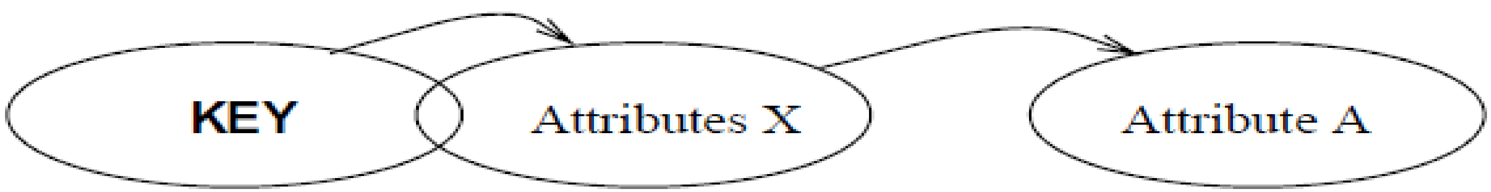


E.g., Reserves has attributes SBDC, (C is for credit card number), the only key is SBD, and we have the FD $S \rightarrow C$.

Then we store the credit card number for a sailor as many times as there are reservations for that sailor.

3NF Violated by X→A: Case 2

- X is not a proper subset of any key.
  - Such X→A is sometimes called a transitive dependency.
  - Because it means there is a chain of FDs $K \rightarrow X \rightarrow A$.

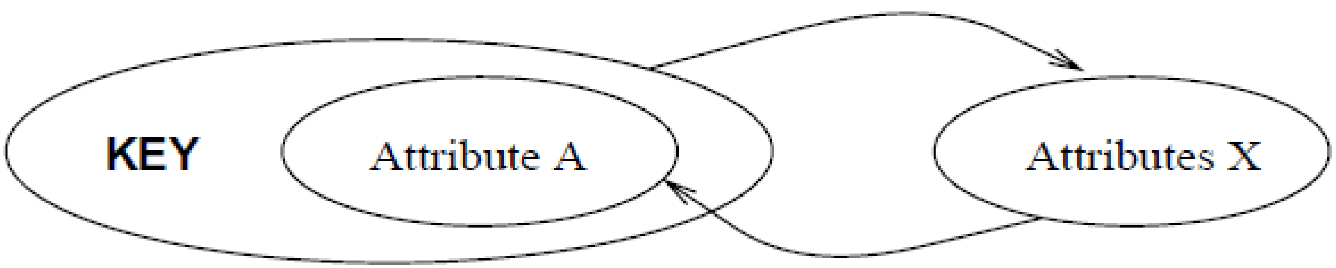


The problem: we cannot associate an X value with a K value, unless we also associate an A value with an X value.

Example: SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$

Redundancy in 3NF

- The problems associated with partial and transitive dependences can persist in 3NF if
  - There is a non-trivial FD X→A,
  - and X is not a superkey,
  - but A is part of a key.



Why 3NF?

- The motivation for 3NF is rather technical.
  - Lossless-join, dependency preserving decomposition does not always exist for BCNF.

  - We can ensure every relation schema can be decomposed into a collection of 3NF relations
    - using only lossless-join, dependency preserving decompositions.

Decomposition into 3NF

- The algorithm for lossless join decomposition into BCNF can be used to obtain a lossless join decomposition into 3NF
  - but does not ensure dependency preservation.
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.

  Problem is that XY may violate 3NF!

  e.g., consider the addition of JPC to `preserve' $JP \rightarrow C$.
  What if we also have $J \rightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - F^+ is equal to G^+.
  - Each FD in G is of the form X→A, where A is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.

A General Algorithm for Calculating Minimal Cover

1. **Put the FDs in a standard form:** Obtain a collection G of equivalent FDs with a single attribute on the right side (using the decomposition axiom).

2. **Minimize the left side of each FD:** For each FD in G, check each attribute in the left side to see if it can be deleted while preserving equivalence to F^+.

3. **Delete redundant FDs:** Check each remaining FD in G to see if it can be deleted while preserving equivalence to F^+.

E. g., Assume F = {A → B, ABCD → E, EF → GH, ACDF → EG}, its minimal cover is:

- A → B, ACD → E, EF → G and EF → H

ACD → E   ACDF → EF   EF → G   EF → E   EF → EG
ACDF → EG

Dependency-Preserving Decomposition into 3NF

- Let R be a relation with a set of F of FDs that is a minimal cover, and let R_1, R_2, …, R_n be a lossless-join decomposition of R.
- Suppose that each R_i is in 3NF, and let F_i denote the projection of F onto the attributes of R_i.
- Do the following:
  - Indentify the set N of FDs in F that are not preserved.
  - For each FD X→A in N, create a relation schema XA and add it to the decomposition of R.
    - *Each XA is in 3NF.*

Proof: *XA is in 3NF*

- Since X→A is in the minimal cover *F*,
  - For any proper subset *Y* of *X*, Y→A does not hold.
  - Therefore, *X* is a key for *XA*.
- For any other FD P→Q that holds over XA
  - *Q* must belong to *X*.

Summary of Schema Refinement

- BCNF: each field contains information that cannot be inferred using only FDs.
  - ensuring BCNF is a good heuristic.
- Not in BCNF? Try decomposing into BCNF relations.
  - Must consider whether all FDs are preserved!
- Lossless-join, dependency preserving decomposition into BCNF impossible? Consider 3NF.
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.