Introduction to Database Systems

2023-Fall

6. Database Design

6.1 Data Dependency and Normalization of Relational Schema

- Some dependent relations exist between attributes.
- ***Function dependency (FD):*** the most basic kind of data dependencies. The value of one or a group attributes can ***decide*** the value of other attributes.

  FD is the most important in general database design.
- ***Multi-valued Dependency (MVD):*** the value of some attribute can decide a group of values of some other attributes.
- ***Join Dependency (JD):*** the constraint of lossless join decomposition.

Function Depedency

- ***Fully Functional Dependency :***
  If X and Y are an attribute set of a relation, Y is fully functional dependent on X, if Y is functionally dependent on X but not on any proper subset of X.
- ***Partial Functional Dependency :***
  A functional dependency X->Y is a partial dependency if Y is functionally dependent on X and Y can be determined by any proper subset of X.
  - For example, we have a relationship AC->B, A->D, and D->B.

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

The Evils of Redundancy

- ***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 → Y holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r, t2 \in r, \pi X(t1) = \pi X(t2)$ implies $\pi Y(t1) = \pi Y(t2)$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- An FD is a statement about ***all*** allowable relations.
  - ***Must be identified based on semantics of application***.
  - Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that K→ R
  - However, K→R does not require K to be *minimal*!

Keys and Functional Dependencies

ID	*name*	*salary*	*dept_name*	*building*	*budget*

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  in_dep (*ID, name, salary, dept_name, building, budget*).

  We expect these functional dependencies to hold:

  dept_name→ *building*

  ID → *building*

  but would not expect the following to hold:

  dept_name → *salary*

Trivial Functional Dependencies

ID	*name*	*salary*	*dept_name*	*building*	*budget*

- *A* functional dependency is ***trivial*** if it is satisfied by all instances of a relation
- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (*ssn, name, lot, rating, hrly_wages*, *hrs_worked*)
- *Notation*: We will denote this relation schema by listing the attributes: SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)
- Some FDs on Hourly_Emps:
  - *ssn* is the key: S → SNLRWH
  - *rating* determines *hrly_wages*: R → W

Example (Contd.)

- Problems due to R →W :
- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Wages

R	W
8	10
5	7

Hourly_Emps2

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

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

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn*→ *did*, *did* → *lot* implies *ssn*→ *lot*
- 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*.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If X⊆Y, then Y→X
  - *Augmentation*: If X→Y, then XZ→YZ for any Z
  - *Transitivity*: If X→Y and Y→Z, then X→Z
- These are *sound* and *complete* inference rules for FDs!

Reasoning About FDs (Contd.)

- Couple of additional rules (that follow from AA):
  - ***Union***: If X → Y and X → Z, then X → YZ
  - ***Decomposition***: If X →YZ, then X→Y and X→Z
- Example: Contracts(*cid, sid, jid, did, pid, qty, value*), and:
  - C is the key: C→CSJDPQV
  - Project purchases each part using single contract: JP → C
  - Dept purchases at most one part from a supplier: SD → P
- JP → C, C → CSJDPQV imply JP → CSJDPQV SD→P implies SDJ→ JP SDJ → JP, JP→ CSJDPQV imply SDJ → CSJDPQV

Reasoning About FDs (Contd.)

- Computing the ***closure*** of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD *X→ Y* is in the closure of a set of FDs *F*. An efficient check:
  - Compute *attribute closure* of X (denoted X+) wrt *F:*
    - Set of all attributes A such that X→ A is in F+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X+
- Does F={A→B, B→C, CD→E} imply A→E?
  - i.e, is A→ E in the closure F+? Equivalently, is E in A+ ?

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- ***Role of FDs in detecting redundancy***:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy here.
    - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value!

1NF

every attribute of a relation must be atomic.

name	dept	address		
		prov	city	street

Non 1NF

name	dept	prov	city	street

1NF

2NF

- R∈1NF and ***no partially function dependency*** exists between attributes.

S(S#, SNAME, AGE, ADDR, C#, GRADE)

--- non 2NF

Problems of non 2NF:

- ***Insert abnormity***

  can not insert the students' information who have not selected course.

- ***Delete abnormity***

  if a student unselect all courses, his basic information is also lost.

- ***Hard to update***

  because of redundancy, it is hard to keep consistency when update.

Resolving:

According to the rule of "**one fact in one place**" to decompose the relation into 2 new relations:

S(S#, SNAME, AGE, ADDR)

SC(S#, C#, GRADE)

3NF

- R∈2NF and ***no transfer (transitive) function dependency*** exists between attributes.

EMP(EMP#, SAL_LEVEL, SALARY)

--- non 3NF

Third Normal Form (3NF)

- Relation R with FDs *F* is in 3NF if, for all X→ A in F+
  - A ∈ X (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

Problems of non 3NF

- ***Insert abnormity***: before the employees's sal_level are decided, the correspondence between sal_level and salary can not input.
- ***Delete abnormity***: if some sal_level has only one man, the correspondence between sal_level and salary of this level will be lost when the man is deleted.
- ***Hard to update***: because of redundancy, it is hard to keep consistency when update.

Resolving:

According to the rule of "**one fact in one place**" to decompose the relation into 2 new relations:

EMP(EMP#,SAL_LEVEL)

SAL(SAL_LEVEL,SALARY)

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if, for all X→ A in F+
  - A ∈ X (called a *trivial* FD), or
  - X contains a key for R.
- ***In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.***
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.

What Does 3NF Achieve?

- If 3NF violated by X → A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs K→ X → A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if relation is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, S→ C, C → S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Material Card

How to define relations to express the information on this card?

Equipment name： Type： Code：							
Unit price： Store place： room rack layer position							
date	voucher No.	coming/going place	take in	take out	balance	sum	remark

Decomposition of a Relation Scheme

- 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 no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

Hourly_Emps (*ssn, name, lot, rating, hrly_wages, hrs_worked*)

- Decompositions should be used only when needed.
  - SNLRWH has FDs S → SNLRWH and R → W
  - Second FD causes violation of 3NF; W values repeatedly
  - associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
  - i.e., we decompose SNLRWH into SNLRH and RW
- The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

- There are three potential problems to consider:
  - ***Some queries become more expensive.***
    - e.g., How much did sailor Joe earn? (salary = W*H)
  - ***Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!***
  - ***Checking some dependencies may require joining the instances of the decomposed relations.***
- ***Tradeoff***: ***Must consider these issues vs. redundancy.***

Summary of Schema Refinement

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless- join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.

Features of Good Relational Designs

- Suppose we combine *instructor* and *department* into *in_dep,* which represents the natural join on the relations *instructor* and *department*

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- There is repetition of information
- Need to use null values (if we add a new department with no instructors)

Decomposition

- The only way to avoid the repetition-of-information problem in the *in_dep* schema is to decompose it into two schemas – instructor and *department* schemas.
- Not all decompositions are good. Suppose we decompose
  employee(ID, name, street, city, salary)

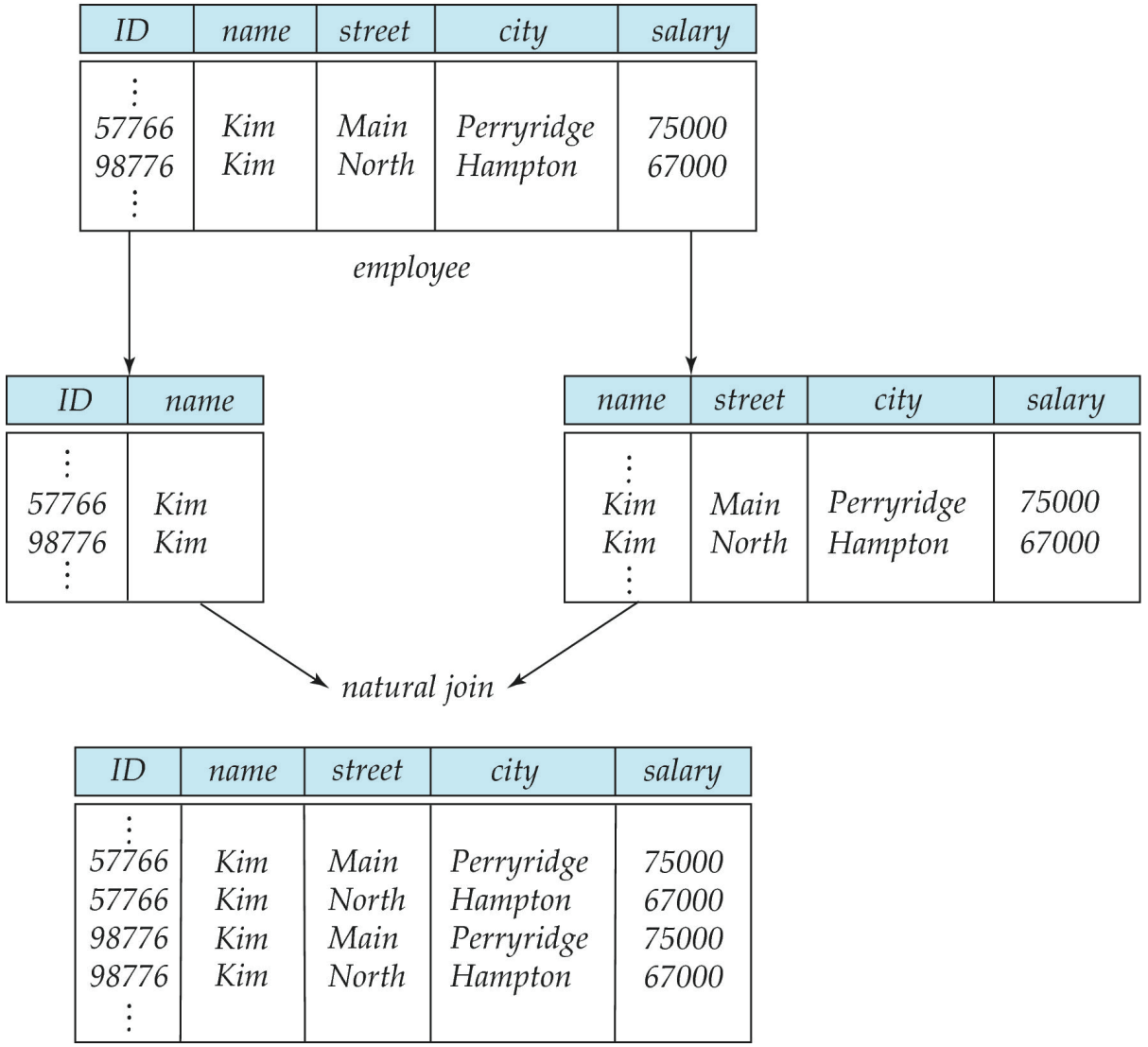
  into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

  The problem arises when we have two employees with the same name
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a ***lossy decomposition.***

A Lossy Decomposition



Lossless Decomposition

- Let *R* be a relation schema and let R_1 and R_2 form a decomposition of R . That is R = $R_1 \cup R_2$
- We say that the decomposition is a **lossless decomposition** if there is no loss of information by replacing R with the two relation schemas $R_1 \cup R_2$
- Formally,

$$\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$

- And, conversely a decomposition is lossy if

$$r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$$

Example of Lossless Decomposition

- Decomposition of $R = (A, B, C)$

  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Boyce-Codd Normal Form

- Example schema that is ***not*** in BCNF:

  in_dep (*ID, name, salary, dept_name, building, budget*)

  because :

  - *dept_name* → *building, budget*
    - holds on *in_dep*
    - but
  - *dept_name* is not a superkey
- When decompose *in_dept* into *instructor* and *department*
  - *instructor* is in BCNF
  - *department* is in BCNF

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving
- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- It is not always possible to achieve both BCNF and dependency preservation
- Consider a schema:

  dept_advisor(s_ID, i_ID, department_name)
- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$
- *dept_advisor* is not in BCNF
  - *i_ID* is not a superkey.
- Any decomposition of *dept_advisor* will not include all the attributes in

  $s\_ID, dept\_name \rightarrow i\_ID$
- Thus, the composition is NOT be dependency preserving

3NF Example

- Consider a schema:

  dept_advisor(s_ID, i_ID, dept_name)

- With function dependencies:

  $i\_ID \rightarrow dept\_name$

  $s\_ID, dept\_name \rightarrow i\_ID$

- Two candidate keys = {*s_ID, dept_name*}, {*s_ID, i_ID* }
- We have seen before that *dept_advisor* is not in BCNF
- *R,* however, is in 3NF
  - *s_ID, dept_name* is a superkey
  - $i\_ID \rightarrow dept\_name$ *and* *i_ID* is NOT a superkey, but:
    - { *dept_name*} – {*i_ID* } = {*dept_name* } and
    - *dept_name* is contained in a candidate key

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF. It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation.
- Disadvantages to 3NF.
  - We may have to use null values to represent some of the possible meaningful relationships among data items.
  - There is the problem of repetition of information.

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

  inst_info (ID, child_name, phone)

  - where an instructor may have more than one phone and can have multiple children
  - Instance of *inst_info*

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

  (99999, David, 981-992-3443)
  (99999, William, 981-992-3443)

Higher Normal Forms

- It is better to decompose *inst_info* into:
  - *inst_child*:

ID	*child_name*
99999	David
99999	William

  - *inst_phone:*

ID	*phone*
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later

Multivalued Dependencies (MVDs)

- Suppose we record names of children, and phone numbers for instructors:
  - *inst_child*(*ID*, *child_name*)
  - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
  - *inst_info*(*ID*, *child_name*, *phone_number*)
  - Example data:
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)
- This relation is in BCNF

Multivalued Dependencies

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \twoheadrightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

MVD -- Tabular representation

- Tabular representation of $\alpha \rightarrow\rightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Example

- In our example:

$$ID \rightarrow\!\!\!\rightarrow child\_name$$
$$ID \rightarrow\!\!\!\rightarrow phone\_number$$

- The above formal definition is supposed to formalize the notion that given a particular value of *Y* (*ID*) it has associated with it a set of values of *Z (child_name)* and a set of values of *W (phone_number)*, and these two sets are in some sense independent of each other.
- Note:
  - If $Y \rightarrow Z$ then $Y \rightarrow\!\!\!\rightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

*Fourth Normal Form

- A relation schema *R* is in **4NF** with respect to a set *D* of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, ***at least one*** of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema *R*
- If a relation is in 4NF it is in BCNF

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \twoheadrightarrow B, B \twoheadrightarrow HI, CG \twoheadrightarrow H \}$
- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition
  a) $R_1 = (A, B)$ (R_1 is in 4NF)
  b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF, decompose into R_3 and R_4)
  c) $R_3 = (C, G, H)$ (R_3 is in 4NF)
  d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF, decompose into R_5 and R_6)
  - $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI$ ➔ $A \twoheadrightarrow HI$, (MVD transitivity), and
  - and hence $A \twoheadrightarrow I$ *(MVD restriction to R_4)*

  e) $R_5 = (A, I)$ (R_5 is in 4NF)
  f) $R_6 = (A, C, G)$ (R_6 is in 4NF)