Schema refinement, Functional dependencies and Normal Form

Kathleen Durant PhD
CS 3200
Lesson 3B

Lecture Outline

- Functional dependency definition
- Schema Refinement
- Redundancy of Data
- Introduction to Normal Form

Functional Dependency

- A functional dependency (FD) has the form X → Y (read X functionally determines Y) where X and Y are sets of attributes in a relation R

> X → Y if and only if:
>
> for any instance r of R
>
> For any tuples t1 and t2 of r
>
> $t_1(X) = t_2(X)$ implies $t_1(Y) = t_2(Y)$

- 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

X → Y iff

any two tuples that agree on X values also agree on Y value

Identifying Functional Dependencies

- FDs are domain knowledge
  - Intrinsic features of the data you're dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of the schema
  - DBMS can only enforce FDs when told about them
- DBMS cannot safely "optimize" FDs either
  - DBMS has only a finite sample of the data
  - An FD constrains the entire domain

Data 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 schema refinements.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: There is no redundancy.
    - Given A → B: Several tuples can have the same A value, and if so, they'll all have the same B value (Redundancy)
- Schema refinement technique: *decomposition* (replace relation ABCD with, say, AB and BCD, or ACD and ABD).

Decomposing Relations

- Decomposition addresses redundancy of data
  - Use FDs to identify "good" ways to split relations
  - Split R into 2+ smaller relations having less redundancy
  - Split up F into subsets which apply to the new relations
- Decomposition should be used judiciously:
  - Is there a reason to decompose a relation?
  - What problems (if any) does the decomposition cause?
- A good decomposition does not :
  - lose information
  - complicate checking of constraints
  - contain anomalies (or at least contains fewer anomalies)

Example: Original Table {S,N,L,R,W,H}

- Social Security #, Name, Lot, Rating, Wage, Hours per week

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

dependency

- FDS S → {S,N,L,R,W,H} AND R → W
- 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

Example Solution

Will 2 smaller tables be better?

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

Set of Functional Dependencies F+

- Informal Definition

  F+ is the set of all FDs logically implied by F

  - Usually F+ is too large to enumerate
  - Some FDs are trivial (EXAMPLE: $A \rightarrow A$)

- Formal Definition

  If F is a set of FDs, then $F+ = \{ X \rightarrow Y \mid F \models X \rightarrow Y \}$

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - *ssn* $\rightarrow$ *did*, *did* $\rightarrow$ *lot* implies *ssn* $\rightarrow$ *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 $\subseteq$ Y, then Y $\rightarrow$ X
  - *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!

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 improve the schema

Reasoning About FDs (Contd.)

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

Closure of FD (Example)

- GIVEN: 1. A → B , 2. B → C and 3. AB → D

- Step 4 if A → B then A → AB (Reflexive & 1)
- Step 5 if A → and B → C then A → C (transitive & 1)
- Step 5 if A→ AB and AB → D then A → ABD (transitive, 3, 4)
- Step 6 if A→ B and B→C then A→ C (1,2,transitivity_
- Step 7 if A→ ABD and A → C then A → ABCD (2, 5, Union)

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
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, not in the SNLRWH example.
- *Tradeoff*: Must consider these issues vs. redundancy.

Normal Form: Codd's Objectives

- Free the collection of relations from undesirable insertion, update and deletion **dependencies**
  - Duplicate data in multiple rows
    - Forced to update/delete all copies of a piece of data
    - How do you know you got all copies of it?
- Reduce the need for restructuring the collection of relations
  - Build an extensible design
- Make the relational model more informative to users
  - Cleaner model should be easier to understand
- Make the collection of relations neutral to the query statistics
  - Designed for general purpose querying

First Normal Form

- Tuples in a relation must contain the same number of fields
- The domain of each attribute is atomic
- The value of each attribute contains only a single value
- No attributes are sets
  - No repeating groups

Levels of Normal Form

- Level 1: No repeating entities or group of elements
  - Do not have multiple columns representing the same type of entity
  - Primary key that represents the entity
- Example: Table mother (MotherName varchar(40), child1 varchar(20), child2(varchar(20)…child8 varchar(20))
  - Create 3 tables: Mother, Children and Offspring
    - Offspring links Mother and Children together

1NF vs. Not 1NF

NOT FIRST NORMAL FORM (1NF) – DUPLICATES ENTITIES					
Mother Id	Mother Name	Child1	Child2	Child3	Child4
1	Elsa	Mary	Alice	NULL	NULL
2	Golda	George	Fred	NULL	NULL
3	Viola	Ava	NULL	NULL	NULL
4	Iris	Kayla	NULL	NULL	NULL
5	Daisy	Harry	NULL	NULL	NULL

- Create Table Mother, Table Offspring and a Table Children
- Link them together via a unique representation (social security number)

Parent Id	Offspring Id
1	11
1	12
2	13
2	14
3	15
4	16
5	17

Mother Id	Mother Name
1	Elsa
2	Golda
3	Viola
4	Iris
5	Daisy

Offspring Id	Offspring Name
11	Mary
12	Alice
13	George
14	Fred
15	Ava
16	Kayla
17	Harry

Benefits of 1NF

- No duplicated data
- Beneficial when you want to extend your database by adding more concepts
- Example: Say you now want to model the father relationship ?
- With the not 1NF solution you are forced to duplicate all of the offspring data in the father relation

Adding the Father Relation

NOT FIRST NORMAL FORM (1NF) – DUPLICATES ENTITIES					
Mother Id	Mother Name	Child1	Child2	Child3	Child4
1	Elsa	Mary	Alice	NULL	NULL
2	Golda	George	Fred	NULL	NULL
3	Viola	Ava	NULL	NULL	NULL
4	Iris	Kayla	NULL	NULL	NULL
5	Daisy	Harry	NULL	NULL	NULL

NOT FIRST NORMAL FORM (1NF) – DUPLICATES ENTITIES					
Father Id	Father Name	Child1	Child2	Child3	Child4
21	Sam	Mary	Alice	Fred	NULL
22	Sal	George	NULL	NULL	NULL
23	Hal	Ava	NULL	NULL	NULL
24	Ed	Kayla	NULL	NULL	NULL
25	George	Harry	NULL	NULL	NULL

- Forced to duplicate child data in both mother and father relationship
- Leads to errors in child data during updates and deletions
- Hard to query child data
- Limits schema
  - 5 children?

1NF with Father Relation

Father Id	Father Name
21	Sam
22	Sal
23	Hal
24	Ed
25	George

Mother Id	Mother Name
1	Elsa
2	Golda
3	Viola
4	Iris
5	Daisy

Parent Id	Offspring Id
1	11
1	12
2	13
2	14
3	15
4	16
5	17
21	11
21	12
21	14
22	13
23	15
24	16
25	17

OneDegree Table contains Mapping between parent and offspring

Offspring Id	Offspring Name
11	Mary
12	Alice
13	George
14	Fred
15	Ava
16	Kayla
17	Harry

Second normal form

- Schema must be in first normal form
  - You have eliminated group sets
  - Every tuple has a unique key
- Each field not in the primary key provides a fact about the entity represented via the (entire) primary key
  - The primary key must be minimal – no extra fields thrown in
  - No partial dependency on part of the primary key
    - Only applies to composite primary key
- Helps you identify a relation that may represent more than one entity
- All fields must be functionally dependent on the complete primary key

Example 2NF vs. Not 2NF

1st Normal Form but NOT 2ndNORMAL FORM				
Mother Id	First Name	Last Name	Hospital	Hospital Address
1	Elsa	General	BIDMC	Boston
2	Golda	Major	MGH	Boston
3	Viola	Funt	TMC	Cambridge
4	Iris	Batter	BIDMC	Brighton
5	Daisy	Mae	Mayo	Allston

2nd NORMAL FORM			
Mother Id	First Name	Last Name	Hospital Id
1	Elsa	General	1
2	Golda	Major	2
3	Viola	Funt	3
4	Iris	Batter	1
5	Daisy	Mae	4

	2nd NORMAL FORM	
Hospital ID	Hospital	Hospital Address
1	BIDMC	Boston
2	MGH	Boston
3	TMC	Cambridge
4	Mayo	Allston

3rd Normal Form

- No dependencies between 2 non-key attributes
- Typically the form most database developers strive to be at

- Bill Kent: Every non-key attribute must provide a fact about the key, the whole key and nothing but the key

Example 3NF vs. Not 3NF

2nd NORMAL FORM				
Mother Id	First Name	Last Name	Hospital Id	Room Number
1	Elsa	General	1	36
2	Golda	Major	2	48
3	Viola	Funt	3	36
4	Iris	Batter	1	41
5	Daisy	Mae	4	32

2nd or 3rd NORMAL FORM		
Hospital ID	Hospital	Hospital Address
1	BIDMC	Boston
2	MGH	Boston
3	TMC	Cambridge
4	Mayo	Allston

3rd NORMAL FORM			
Mother Id	First Name	Last Name	Registration Id
1	Elsa	General	1
2	Golda	Major	2
3	Viola	Funt	3
4	Iris	Batter	4
5	Daisy	Mae	5

3rd Normal Form		
Registration Id	Hospital Id	Room Id
1	1	36
2	2	48
3	3	36
4	1	41
5	4	32

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 contains a key for R, or
  - A is part of some key for R. (Relaxation from BCNF)
- *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 BCNF not achievable (e.g., no ``good'' decomposition, or performance considerations).

What Does 3NF Achieve?

- If 3NF is violated by X $\rightarrow$ A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly in the relation
  - X is not a proper subset of any key.
    - There is a chain of FDs K $\longrightarrow$ X $\longrightarrow$ 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 $\longrightarrow$ C, C $\longrightarrow$ 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.

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.

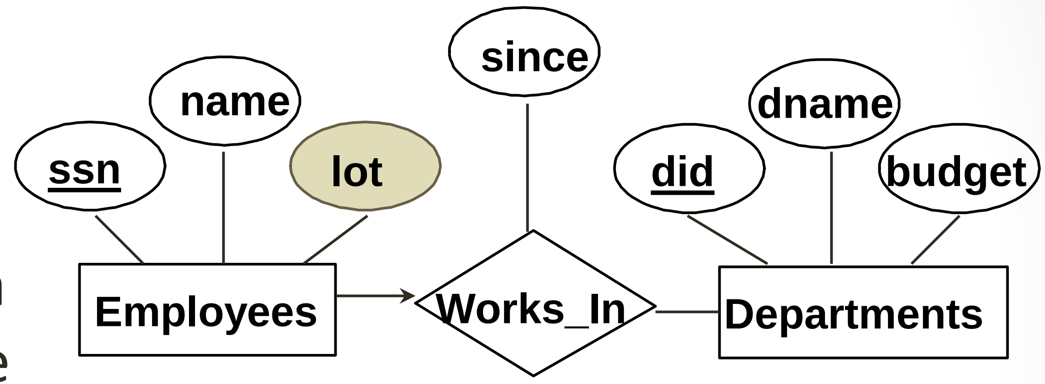
Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs S $\rightarrow$SNLRWH and R $\rightarrow$ 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?

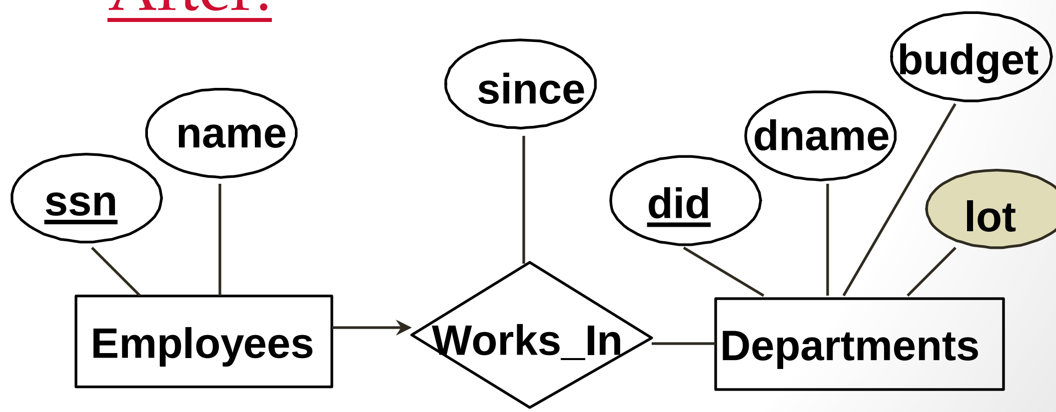
Refining an ER Diagram

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

Before:



After:



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 contains a key for the relation 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.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

Normal Form Tips

- Review your attributes in your tables and ensure that they are facts about the complete key and only the complete key
- No duplicating groups in a table
- Split many to many relationships up into 2 many to 1 relationships by identifying the relation that maps them together

Example

- Students takes Courses M-to-M relationship
  - Many students to a Course
  - Many courses to a Student
- Represent using 2 M-to-1 relationships
  - Students has an Enrollment M-to-1
  - Enrollment in a Class 1-to-M

Student Table
<u>StudentID</u>

Class Table
<u>ClassID</u>

Enrollment
<u>StudentId, ClassId</u>

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 all decompositions that exists lead to a loss of information then should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.