Schema Refinement and Normalization

COMP 3380 - Databases: Concepts and Usage

Department of Computer Science
The University of Manitoba
Fall 2018

Schema Refinement

Review: Database Design Process

1. Requirements collection & analysis
2. Conceptual DB design (ER model)
3. Logical design (data model mapping, e.g., map ER to tables)
4. **Schema refinement (e.g., normalization)**
5. Physical design
6. System implementation & tuning (e.g., application & security design)

The Evils of Redundancy

- **Redundancy** is at the root of several problems associated with relational schemas:
  - **Redundant storage**
    - Some info is stored repeatedly
  - **Insertion anomalies**
    - May not be possible to store certain info unless some other (unrelated) info is stored as well
  - **Deletion anomalies**
    - May not be possible to delete certain info without losing some other (unrelated) info is stored as well
  - **Update anomalies**
    - If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated

Example: Redundancy

eID	eName	rank	hrlyWages	hrsWorked
123	Albert	8	10	40
131	Bob	5	7	30
231	Carl	8	10	30
434	Don	5	7	32
612	Ed	8	10	40

- Suppose (i) eID is a candidate key & (ii) rank determines hrlyWages

- Redundant Storage
  - "rank=8 corresponds to hrlyWages=10" is repeated 3 times
- Insertion anomalies
  - Cannot record hrlyWages for rank=6 unless there exists a rank-6 emp
  - Cannot insert an employee tuple unless we know his rank/hrlyWages (or we put NULL values)

Example: Redundancy

eID	eName	rank	hrlyWages	hrsWorked
123	Albert	8	10	40
131	Bob	5	7	30
231	Carl	8	10	30
434	Don	5	7	32
612	Ed	8	10	40

- Suppose (i) eID is a candidate key & (ii) rank determines hrlyWages
- Deletion anomalies
  - If we delete all employee tuples with rank=5, we lose the information about hrlyWages for rank=5
- Update anomalies
  - hrlyWages in the 1st tuple could be updated without making a similar change in the 3rd tuple ➔ inconsistency

The Evils of Redundancy

- **Functional dependencies (FDs)**: Integrity constraints that can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: **Decomposition**
  - **Splits a table into *many* tables, each with *fewer* attributes**
    - E.g., replace *R (A,B,C,D)* with *R1 (A,B)* and *R2 (B,C,D)*
  - Should be used judiciously: **Wrong decomposition may lose information!**

Functional Dependencies

- **{A, B, C} → D**
  - A,B,C together ***determine*** D; so, A,B,C is a **determinant**
  - D is said to *depend* on A,B,C
  - Sometimes written as **A,B,C → D** or **ABC → D**
- FDs are a special kind of integrity constraint
- We are most interested in cases where there is a single attribute on the RHS
- The most uninteresting cases are the ***trivial cases***:
  - E.g., ABC → A

Functional Dependencies

- A **functional dependency (FD)**

  X→Y

  holds over relation R *if*, for **every** allowable instance *r* of R & every two tuples *t1, t2* in *r*

  if *t1*.X = *t2*.X, then *t1*.Y = *t2*.Y
- Given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes)
- Informally, ***precisely one*** Y-value is associated with each X value

Example

X	Y	Z
1	2	4
1	3	4

- It is *possible* (but *not necessary*) that
  - X → Z, Y → X, Y → Z, Z → X
- It is *not* the case that X → Y
- It is *not* the case that Z → Y

Example

- $X \rightarrow Z$ holds for the above instance, but *not* necessarily hold for *all* instances
- Similar comments for $(Y \rightarrow X)$, $(Y \rightarrow Z)$, and $(Z \rightarrow X)$

- $X \rightarrow Y$ does *not* hold because $(t1.X = t2.X)$ but $(t1.Y \neq t2.Y)$
- Similarly, $Z \rightarrow Y$ does *not* hold because $(t1.Z = t2.Z)$ but $(t1.Y \neq t2.Y)$

Instantiated Example

X	Y	Z
1	2	4
1	3	4
- It is *possible* (but *not necessary*) that
  - X→Z,
  - Y→X
  - Y→Z,
  - Z→X
- It is *not* the case that
  - X→Y,
  - Z→Y

rank	eName	hrlyWages
5	Bob	$7
5	Don	$7
- It is *possible* (but *not necessary*) that
  - rank → hrlyWages,
  - eName → rank,
  - eName → hrlyWages,
  - hrlyWages → rank
- It is *not* the case that
  - rank → eName,
  - hrlyWages → eName

Functional Dependencies

- An FD is a statement about ***all*** allowable instances
  - Must be identified based on semantics of application
  - Given some allowable instance *r1* of R:
    - we ***can check if it violates some FDs***, but
    - we ***cannot tell if the FD holds over R***
- K is a **superkey** for R means that **K → attrs(R)**
  - Note: **K is not required to be minimal**

Reference: Reasoning about FDs

- Given some FDs, we can usually infer additional FDs
  - E.g., ($eID \rightarrow dID$) & ($dID \rightarrow addr$) implies ($eID \rightarrow addr$)
- 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

Reference: Reasoning about FDs: Dependency Closure vs. Attribute Closure

- **Dependency closure** F^+ = the set of all FDs that are implied by a set of FDs F
  - E.g., { ($eID \rightarrow dID$), ($dID \rightarrow addr$) }$^+$
    = { ($eID \rightarrow dID$), ($dID \rightarrow addr$), ($eID \rightarrow addr$) }
- **Attribute closure** X^+ = the set of all attrs that are implied by a set of attrs X wrt F
  - E.g., { eID }$^+$ = { eID, dID, $addr$ }
  - E.g., { dID }$^+$ = { dID, $addr$ }
  - E.g., { eID, dID }$^+$ = { eID, dID, $addr$ }

Reasoning about FDs: Armstrong's Axioms

- For X, Y, Z are sets of attributes:
  - **Reflexivity:** If $Y \subseteq X$, then $X \rightarrow Y$
  - **Augmentation:** If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for all Z
  - **Transitivity:** If ($X \rightarrow Y$) and ($Y \rightarrow Z$), then $X \rightarrow Z$
- These are *sound* and *complete* inference rules for FDs

Reasoning about FDs: Additional Rules

- For X, Y, Z are sets of attributes:
  - **Union:** If $(X \rightarrow Y)$ and $(X \rightarrow Z)$, then $X \rightarrow YZ$
  - **Decomposition:** If $X \rightarrow YZ$, then $(X \rightarrow Y)$ and $(X \rightarrow Z)$
- These additional rules can be derived from Armstrong's Axioms

Example 1

- Prove: If ($X \rightarrow Y$) and ($WY \rightarrow Z$), then $WX \rightarrow Z$

1. $X \rightarrow Y$	given (FD1)
2. $WX \rightarrow WY$	1, augmentation (W)
3. $WY \rightarrow Z$	given (FD2)
4. $WX \rightarrow Z$	2, 3, transitivity

- This additional rule is called **pseudo-transitivity rule**

Example 2

- Disprove: If (X → Z) and (Y → Z), then X → Y

  Counterexample:

X	Y	Z
1	2	4
1	3	4

Example 3

- Given the following FDs

  1. s# → sName, city 2. city → status

  3. p# → pName 4. s#, p# → qty

  show (s#, p#) is a candidate key of *SPJ* (*s#, p#, status, city, qty*)

- Proof: First prove that (s#, p#) is a superkey

1.	s# → city	FD1, decomposition
2.	s# → status	1, FD2, transitivity
3.	s# → s#	reflexivity
4.	s# → s#, status, city	1, 2, 3, union
5.	s#, p# → s#, p#, status, city	4, augmentation (p#)
6.	s#, p# → s#, p#, status, city, qty	5, FD4, union

  Then prove (s#, p#) is a minimal superkey

Example 3 (Cont'd)

- Given the following FDs

  1. s# → sName, city   2. city → status

  3. p# → pName   4. s#, p# → qty

  show (s#, p#) is a candidate key of *SPJ* (*s#, p#, status, city, qty*)

- <u>Proof</u>: After proving that (s#, p#) is a superkey, prove (s#, p#) is a candidate key (i.e., minimal superkey)
  1. Can s# be a superkey? **NO**
     - E.g., s# → p# does not hold (because p# does not appear on the RHS of any FD)
  2. Can p# be a superkey? **NO**
     - E.g., p# → s# does not hold (because s# does not appear on the RHS of any FD)

  <u>Note:</u> (s#, p#) is the *only* candidate key → (s#, p#) is the primary key

Example 4

- Consider R (A,B,C,D,E) which satisfies the following FDs:

  1. AB $\rightarrow$ C 2. B $\rightarrow$ D 3. D $\rightarrow$ E

  Explain why A is not a candidate key of R.

- Show: A is not a candidate key of R

  Since B does not appear on the RHS of any FDs, A $\rightarrow$ B does not hold. So, A cannot be a superkey of R, and hence A is not a candidate key of R.

  Note: It is also *not* the case that (A $\rightarrow$ C), (A $\rightarrow$ D), or (A $\rightarrow$ E).

Example 5

- Consider R (A,B,C,D,E) which satisfies the following FDs:

  1. AB → C   2. B → D   3. D → E

  Explain why ABD is not a candidate key of R.

- <u>Show: ABD is not a candidate key of R</u>

  Since AB is a superkey of R (see the proof below), ABD is not a candidate key of R.

1.	AB → A	reflexivity
2.	AB → B	reflexivity
3.	AB → D	2, FD2, transitivity
4.	AB → E	3, FD3, transitivity
5.	AB → ABCDE	1, 2, FD1, 3, 4, union

Normalization

Normal Forms

- "Whether any schema refinement is needed?"
  - If a relation is in a certain **normal form** (e.g., 3NF, BCNF, 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
  - E.g., consider a relation R with 3 attributes, ABC.
    - If no FDs hold, there is no redundancy here.
    - If A → B, several tuples having the same A value will all have the same B value.
- **Normalization:** The process of removing redundancy from data

1NF (First Normal Form)

- A relation is in **1NF** if every attribute contains only **atomic** values (i.e., **single-valued** attribute, *no* lists or sets)
  - Tables cannot have 2^+ entries for the same cell
  - E.g., cannot enter "Elmasri & Navathe" in the same cell for "author"
    ➔ *Books* (<u>*ISBN*</u>, *bookName, authors*) is **not in 1NF** because attribute *authors* contains a list of author names
  - E.g., *Emp0* (<u>*empID*</u>, *empName, childrenNames*) is **not in 1NF** because attribute *childrenNames* contains a list of children names
  - E.g., *SPJ0* (<u>*s#*</u>, *p#list, status, city, totalQty*) is **not in 1NF** because of the multiple values for attribute *p#list*

Example: 1NF

- *Books* (*ISBN*, *bookName, authors*) is **not in 1NF**
  → normalize into
  - *Books* (*ISBN*, *bookName*) **1NF**
  - *BookAuthors* (*ISBN, author*) **1NF**

- *Emp0* (*empID*, *empName, childrenNames*) is **not in 1NF**
  → normalize into
  - *Emp1* (*empID*, *empName*) **1NF**
  - *EmpChildren* (*empID, childName*) **1NF**

2NF (Second Normal Form)

- A relation R is in **2NF** if:
  1. R is in 1NF, and
  2. every attribute A in R *either* appears in a candidate key *or* is not partially dependent on a candidate key (i.e., **no partial key dependency**)
- A functional dependency **XY→Z** is called a **partial dependency** if there is a *proper subset* $Y \subset XY$ such that Y→Z (i.e., Z is partially dependent on XY)

Example: 2NF

- *SPJ1* (<u>*s#, p#*</u>, *status, city, qty*) with 3 FDs:
  1. *s#* ➔ *city*
  2. *city* ➔ *status*
  3. *s#, p#* ➔ *qty*

  is in 1NF, but **not in 2NF** (because *city* is partially dependent on {*s#, p#*} due to FD1 "*s#*➔*city*")

  ➔ normalize into 2 relations:

Supplier2 (<u>*s#*</u>, *status, city*)	**2NF**
SP (<u>*s#, p#*</u>, *qty*)	**2NF**

Example: 2NF (Details)

- *Supplier2* (<u>*s#*</u>, *status, city*) **2NF**
  - *Supplier2* is in 1NF, and
  - 3 attributes in *Supplier2*: *s#, status, city*
    - *s#* appears in a candidate key,
    - *status* is not partially dependent on a candidate key,
    - *city* is not partially dependent on a candidate key.
- *SP* (<u>*s#, p#*</u>, *qty*) **2NF**
  - *SP* is in 1NF, and
  - 3 attributes in *SP*: *s#, p#, qty*
    - *s#* appears in a candidate key,
    - *p#* appears in a candidate key,
    - *qty* is not partially dependent on a candidate key.

3NF (Third Normal Form)

- A relation R is in **3NF** if:
  1. R is in 2NF, and
  2. for every functional dependency X→A, one of the following conditions hold:
     i. A is part of some candidate keys for R, *or*
     ii. A ∈ X (i.e., a trivial FD), *or*
     iii. X is a superkey

     (i.e., **no partial dependency** & **no transitive dependency**)

3NF (Third Normal Form)

- A functional dependency **X→Z** is called a **transitive dependency** if there is an attribute set Y (which is *not* a subset of *any* candidate keys) such that
  i. (X → Y) and (Y → Z) hold, but
  ii. Z → X does not hold

  (i.e., Z is transitively dependent on X, thro' the chain of X→Y→Z)

Example: 3NF

- *Supplier2* (<u>*s#*</u>, *status*, *city*) with 2 FDs:
  1. *s#* → *city*
  2. *city* → *status*

  is in 2NF, but **not in 3NF** (because *status* is transitively dependent on *s#*)

  → normalize into 2 relations:

Supplier3 (<u>*s#*</u>, *city*)	**3NF**
CityInfo (<u>*city*</u>, *status*)	**3NF**

- *SP* (<u>*s#*, *p#*</u>, *qty*) is in 2NF & **3NF**

Example: 3NF (Details)

- *Supplier3* (<u>*s#*</u>, *city*)   **3NF**
  - *Supplier3* is in 2NF, and
  - 1 relevant FD (*s#*→*city*):   *s#* is a superkey.

- *CityInfo* (<u>*city*</u>, *status*)   **3NF**
  - *CityInfo* is in 2NF, and
  - 1 relevant FD (*city*→*status*):   *city* is a superkey.

- *SP* (<u>*s#, p#*</u>, *qty*)   **3NF**
  - *SP* is in 2NF, and
  - 1 relevant FD (*s#*,*p#*→*qty*):   {*s#*, *p#*} is a superkey.

BCNF (Boyce-Codd Normal Form)

- A relation R is in **BCNF** if for every functional dependency $X \rightarrow A$ that holds over R, one of the following is true:
  i. $A \in X$ (i.e., a trivial FD), or
  ii. X is a superkey

  (i.e., **all determinants X must be superkeys**)

Example: BCNF

- With 3 FDs:
  1. *s#* ➔ *city*
  2. *city* ➔ *status*
  3. *s#, p#* ➔ *qty*

  Supplier3 (*s#*, *city*) is in 3NF & **BCNF**

  CityInfo (*city*, *status*) is in 3NF & **BCNF**

  SP (*s#, p#*, *qty*) is in 3NF & **BCNF**

Example: BCNF (Details)

- *Supplier3* (<u>*s#*</u>, *city*) **BCNF**
  - 1 relevant FD (s#➔city): *s#* is a superkey.
- *CityInfo* (<u>*city*</u>, *status*) **BCNF**
  - 1 relevant FD (*city*➔*status*): *city* is a superkey.
- *SP* (<u>*s#, p#*</u>, *qty*) **BCNF**
  - 1 relevant FD (*s#*,*p#*➔*qty*): {*s#*,*p#*} is a superkey.

Summary of Normal Forms

- **Non-key attribute** (aka non-prime attribute)
  = An attribute that is *not* part of any candidate keys

- **1NF:** Every attribute contains only **atomic** values

- **2NF:** 1NF + Every attribute A in R *either* appears in a candidate key *or* is not partially dependent on a candidate key
  - No non-key attribute is partially dependent on some candidate keys
    (i.e., **no partial key dependency**)

Summary of Normal Forms

- **3NF:** 2NF + For every FD X→A, one of the following conditions hold: (i) A is part of some candidate keys, *or* (ii) $A \in X$, *or* (iii) X is a superkey
  - No non-key attribute is transitively dependent on some candidate keys

    (i.e., **no partial dependency** & **no transitive dependency**)
- **BCNF:** For every FD X→A that holds over R, one of the following is true: (i) $A \in X$, or (ii) X is a superkey
  - **All determinants X must be superkeys**

Normalization & Design: Comments

- There are too many anomalies (problems) with 1NF and 2NF ➔ most organizations go to 3NF or better
  - E.g., With *SPJ1 (<u>s#, p#,</u> status, city, qty)* in 1NF,
    - cannot record facts about suppliers that do not currently supply any parts
    - cannot preserve facts about suppliers when deleting the last tuple related to the suppliers
  - E.g., With *Supplier2 (<u>s#,</u> status, city)* in 2NF,
    - cannot record status about cities where no supplier resides
    - cannot preserve status about cities when deleting the last supplier in the cities

Normalization & Design: Comments

- **If the primary key consists of only 1 attribute, the relation is automatically in 2NF**
  - E.g., *Supplier2 (s#, status, city)*
- 3NF is generally *not* satisfactory if the following are all true:
  - The relation has multiple candidate keys
  - The candidate keys are composite (i.e., consist of 2^+ attributes)
  - The candidate keys overlap
- **If a relation has only 2 attributes, it is automatically in BCNF**
  - E.g., *Supplier3 (s#, city)*

Normalization & Design: Comments

- If R is in 3NF, some redundancy is possible
- Compromise: Used when BCNF *not* achievable (e.g., no "good" decomposition exists, or when there are performance considerations that warrant 3NF)
- Note: BCNF drops the condition "A is part of some keys for R for every FD X → A"

  i.e., if R is in BCNF, then X is a superkey for every functional dependency X → A

Normalization & Design: Comments

- 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.
  - Decompositions should be carried out and/or re-examined (while keeping *performance requirements* in mind)
- <u>Other NFs:</u> **4NF**, **5NF** (aka PJNF), **DKNF**

Normalization & Design: Comments