CompSci 516
Data Intensive Computing Systems

Lecture 10
Normalization

Instructor: Sudeepa Roy

Announcements

- Change in time of Sudeepa's office hour (only for) next week
  - 11:45 am to 12:45 pm – Monday 10/3
- Feedback on project proposal posted on sakai
- Midterm syllabus: up to Lecture 10
  - We will start a new topic Transactions next week

Where are we now?

We learnt

- ✓ Relational Model and Query Languages
  - ✓ SQL, RA, RC
  - ✓ Postgres (DBMS)
  - HW1
- ✓ Map-reduce and spark
  - HW2
- ✓ DBMS Internals
  - ✓ Storage
  - ✓ Indexing
  - ✓ Query Evaluation
  - ✓ Operator Algorithms
  - ✓ External sort
  - ✓ Query Optimization

Next

- Database Normalization
  - (for good schema design)
- Transactions
  - Basic concepts
  - Concurrency control
  - Recovery

Reading Material

- Database normalization
  - [RG] Chapter 19.1 to 19.5, 19.6.1, 19.8 (overview)
  - [GUW] Chapter 3

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

What will we learn?

- What goes wrong if we have redundant info in a database?
- Why and how should you refine a schema?
- Functional Dependencies – a new kind of integrity constraints (IC)
- Normal Forms
- How to obtain those normal forms

Example

The list of hourly employees in an organization

<u>ssn (S)</u>	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN

Example

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- key = SSN
- Suppose for a given rating, there is only one hourly_wage value
- Redundancy in the table
- Why is redundancy bad?

Why is redundancy bad?

The list of hourly employees in an organization

<u>ssn (S)</u>	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

1. Redundant storage:
   - Some information is stored repeatedly
   - The rating value 8 corresponds to hourly_wage 10, which is stored three times

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10 → 9	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

2. Update anomalies
   - If one copy of data is updated, an inconsistency is created unless all copies are similarly updated
   - Suppose you update the hourly_wage value in the first tuple using UPDATE statement in SQL -- inconsistency

Why is redundancy bad?

The list of hourly employees in an organization

<u>ssn (S)</u>	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

3. Insertion anomalies:
   - It may not be possible to store certain information unless some other, unrelated info is stored as well
   - We cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value

Why is redundancy bad?

The list of hourly employees in an organization

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

4. Deletion anomalies:
   - It may not be possible delete certain information without losing some other information as well
   - If we delete all tuples with a given rating value (Attishoo, Smiley, Madayan), we lose the association between that rating value and its hourly_wage value

Nulls may or may not help

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

- Does not help redundant storage or update anomalies
- May help insertion and deletion anomalies
  - can insert a tuple with null value in the hourly_wage field
  - but cannot record hourly_wage for a rating unless there is such an employee (SSN cannot be null) – same for deletion

Summary: Redundancy

Therefore,

- Redundancy arises when the schema forces an association between attributes that is "not natural"
- We want schemas that do not permit redundancy
  - at least identify schemas that allow redundancy to make an informed decision (e.g. for performance reasons)
- Null value may or may not help

- Solution?
  - decomposition of schema

Decomposition

ssn (S)	name (N)	lot (L)	rating (R)	hourly-wage (W)	hours-worked (H)
111-11-1111	Attishoo	48	8	10	40
222-22-2222	Smiley	22	8	10	30
333-33-3333	Smethurst	35	5	7	30
444-44-4444	Guldu	35	5	7	32
555-55-5555	Madayan	35	8	10	40

ssn (S)	name (N)	lot (L)	rating (R)	hours-worked (H)
111-11-1111	Attishoo	48	8	40
222-22-2222	Smiley	22	8	30
333-33-3333	Smethurst	35	5	30
444-44-4444	Guldu	35	5	32
555-55-5555	Madayan	35	8	40

rating	hourly_wage
8	10
5	7

Decompositions should be used judiciously

1. Do we need to decompose a relation?
   - Several normal forms
   - If a relation is not in one of them, may need to decompose further

2. What are the problems with decomposition?
   - Lossless joins, Dependency preservations (soon)
   - Performance issues -- decomposition may both
     - help performance (for updates, some queries accessing part of data), or
     - hurt performance (new joins may be needed for some queries)

Functional Dependencies (FDs)

- A functional dependency (FD) X → Y holds over relation R if, for every allowable instance *r* of R:
  - 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
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

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

What is an FD here?

Functional Dependencies (FDs)

- A functional dependency (FD) $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - 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
  - $t1 \in r,\ t2 \in r,\ \Pi_X(t1) = \Pi_X(t2)$ implies $\Pi_Y(t1) = \Pi_Y(t2)$

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

What is an FD here?

$AB \rightarrow C$

Note that, AB is not a key

not a correct question though.. see next slide!

Functional Dependencies (FDs)

- 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
  - assume R = all attributes of R too
  - However, S →R does not require S to be minimal
  - e.g. S can be a superkey

Example

- Consider relation obtained from Hourly_Emps:
  - Hourly_Emps (ssn, name, lot, rating, hourly_wage, hours_worked)

- Notation: We will denote a relation schema by listing the attributes: SNLRWH
  - Basically the set of attributes {S,N,L,R,W,H}

- FDs on Hourly_Emps:
  - ssn is the key: S → SNLRWH
  - rating determines hourly_wages: R → W

Armstrong's Axioms

- X, Y, Z are sets 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 $X \rightarrow Z$

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

Apply these rules on $AB \rightarrow C$ and check

Armstrong's Axioms

- X, Y, Z are sets 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
  - sound: then only generate FDs in F^+ for F
  - complete: by repeated application of these rules, all FDs in F^+ will be generated

Additional Rules

- Follow from Armstrong's Axioms

- Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

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

$A \rightarrow B$, $A \rightarrow C$

$A \rightarrow BC$

$A \rightarrow BC$

$A \rightarrow B$, $A \rightarrow C$

Closure of a set of FDs

- Given some FDs, we can usually infer additional FDs:
  - SSN → DEPT, and DEPT → 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*

To check if an FD belongs to a closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure can be exponential in #attributes

- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F

- No need to compute F^+

- Compute attribute closure of X (denoted X^+) wrt F:
  - Set of all attributes A such that $X \rightarrow A$ is in F^+

Computing Attribute Closure

Algorithm:

- closure = X
- Repeat until no change
  - if there is an FD $U \rightarrow V$ in F such that $U \subseteq$ closure, then closure = closure $\cup$ V
- Check if Y is in X^+

- Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\,D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Normal Forms

- Question: given a schema, how to decide whether any schema refinement is needed at all?
- If a relation is in a certain normal forms, it is known that certain kinds of problems are avoided/minimized
- Helps us decide whether decomposing the relation is something we want to do

FDs play a role in detecting redundancy

Example

- Consider a relation R with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given A → B: Several tuples could have the same A value, and if so, they'll all have the same B value – redundancy – decomposition may be needed if A is not a key

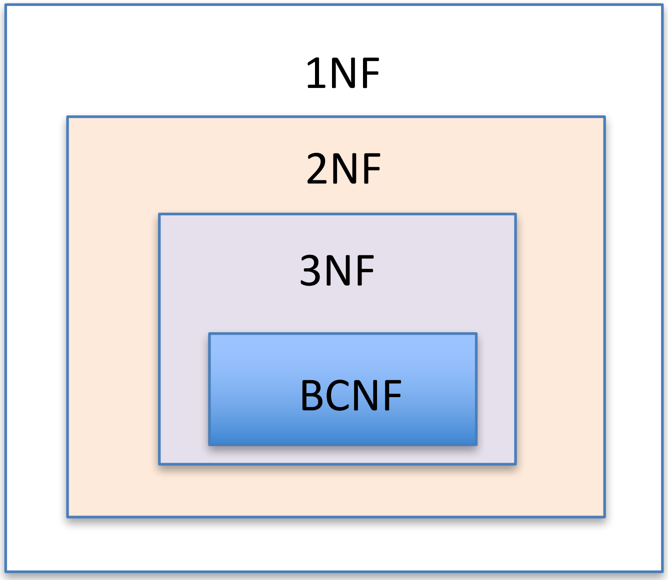
Normal Forms

R is in BCNF

$\Rightarrow$ R is in 3NF

$\Rightarrow$ R is in 2NF (a historical one, not covered)

$\Rightarrow$ R is in 1NF (every field has atomic values)



Definitions next

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs *F* is in BCNF if, for all X → A in F
  - A ϵ X (called a trivial FD), or
  - X contains a key for R
    - i.e. X is a superkey

Observations: BCNF

R is in BCNF if the only non-trivial FDs that hold over R are key constraints

- each tuple has a key and a bunch of other attributes
- 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

- Suppose X → A and the relation is in BCNF – what can you infer?

- The two tuples must be identical (assuming a set this relation is not possible)
  - otherwise, X is not the key
  - and X → A is a non-trivial F.D.
  - violated BCNF

X	Y	A
x	y1	a
x	y2	?

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.

two conditions for BCNF

- Minimality of a key is crucial in third condition in 3NF
  - every attribute is part of some superkey (= set of all attributes)

BCNF vs. 3NF

- If R is in BCNF, obviously in 3NF
- If R is in 3NF, some redundancy is possible
  - when X → A and A is part of a key (not allowed in BCNF)

- Example:
  - Reserves(<u>S, B, D</u>, C), C = credit card, S → C and C → S
  - Since <u>SBD</u> is a key, <u>CBD</u> is also a key, 3NF not violated
  - but in all tuples recording the same S value, the same (S, C) pair is redundantly recorded
  - note: relation is not in BCNF since both S and C are not superkeys

Decomposition of a Relation Schema

- Consider relation R contains attributes A1 ... An

- A decomposition of R consists of replacing R by two or more relations such that “no attribute is lost” and “no new attribute appears”, i.e.
  - Each new relation schema contains a subset of the attributes of R
  - Every attribute of R appears as an attribute of one of the new relations
  - E.g., Can decompose SNLRWH into SNLRH and RW

- What are the potential problems with an arbitrary decomposition?

Good properties of decomposition

- Lossless join decomposition
- Dependency preserving decomposition

Lossless Join Decompositions

- 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 $\pi_X(r) \bowtie \pi_Y(r) \supseteq r$
- In general, the other direction does not hold
  - If it does, the decomposition is lossless-join

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

- Decompose into SP and PD -- is the decomposition lossless?
- How about SP and SD?

For lossless decomposition of R into R1, R2
- either R1 ∩ R2 → R1
- or R1 ∩ R2 → R2

Dependency Preserving Decomposition

- Consider <u>C</u>SJDPQV, C is key, JP → C and SD → P
  - Lossless decomposition: CSJDQV and SDP
    - SD key of (SDP)!
  - Problem: Checking JP → C requires a join

- Dependency preserving decomposition:
  - join is not needed to check a dependency

Algorithm: Decomposition into BCNF

- Input: relation R with FDs F

If $X \rightarrow Y$ violates BCNF, decompose R into R - Y and XY.

Repeat until all new relations are in BCNF w.r.t. the given F

- Gives a collection of relations that are
  - in BCNF
  - lossless join decomposition
  - and guaranteed to terminate
  - but a dependency-preserving decomposition may not exist (example in book)

Decomposition into BCNF (example)

- CSJDPQV, key C, F = {JP → C, SD → P, J → S}
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV

- Note:
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions

Other kinds of dependencies and normal forms

- Multi-valued dependencies
- Join dependencies
- Inclusion dependencies
- 4NF, 5NF
- See book if interested (not covered in class)

Summary

- Redundancy is not desired typically
  - not always, mainly due to performance reasons
- Functional dependencies – capture redundancy
- Decompositions – eliminate dependencies
- Normal forms
  - Guarantees certain non-redundancy
  - BCNF and 3NF
- Lossless join and dependency-preserving joins
- How to decompose into BCNF