CompSci 516
Data Intensive Computing Systems

Lecture 5
Design Theory and Normalization

Instructor: Sudeepa Roy

Announcements

- HW1 deadline:
  - Due on 09/21 (Thurs), 11:55 pm, no late days
- Project proposal deadline:
  - Preliminary idea and team members due by 09/18 (Mon) by email to the instructor
  - Proposal due on sakai by 09/25 (Mon), 11:55 pm

Today

- Finish RC from Lecture 4
  - DRC
  - More example
- Normalization

DRC: example

Sailors(sid, sname, rating, age)
Boats(bid, bname, color)
Reserves(sid, bid, day)

- Find the name and age of all sailors with a rating above 7

TRC:
{P | ∃ S ϵ Sailors (S.rating > 7 ∧ P.name = S.name ∧ P.age = S.age)}

DRC:
{<N, A> | ∃ <I, N, T, A> ϵ Sailors ∧ T > 7}

- Variables are now domain variables
- We will use use TRC
  - both are equivalent

More Examples: RC

- The famous "Drinker-Beer-Bar" example!

UNDERSTAND THE DIFFERENCE IN ANSWERS FOR ALL FOUR DRINKERS

Acknowledgement: examples and slides by Profs. Balazinska and Suciu, and the [GUW] book

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 1

Find drinkers that frequent some bar that serves some beer they like.

Q(x) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

a shortcut for
{x | ∃Y ϵ Frequents Z ϵ Serves W ϵ Likes (T.drinker = x.drinker ∧ T.bar = Z.bar ∧ W.beer = ……}

The difference is that in the first one, one variable = one attribute
in the second one, one variable = one tuple (Tuple RC)
Both are equivalent and feel free to use the one that is convenient to you

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Drinker Category 2/3/4

Find drinkers that frequent some bar that serves some beer they like.

Q(x) = ∃y. ∃z. Frequents(x, y)∧Serves(y,z)∧Likes(x,z)

Find drinkers that frequent only bars that serves some beer they like.

Q(x) =

Find drinkers that frequent some bar that serves only beers they like.

Q(x) =

Find drinkers that frequent only bars that serves only beer they like.

Q(x) =

Why should we care about RC

- RC is declarative, like SQL, and unlike RA (which is operational)
- Gives foundation of database queries in first-order logic
  - you cannot express all aggregates in RC, e.g. cardinality of a relation or sum (possible in extended RA and SQL)
  - still can express conditions like "at least two tuples" (or any constant)
- RC expression may be much simpler than SQL queries
  - and easier to check for correctness than SQL
  - power to use ∀ and ⇒
  - then you can systematically go to a "correct" SQL query

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer (so much) that they frequent all bars that serve it

Q(x) = ∃y. Likes(x, y)∧∀z.(Serves(z,y) ⇒ Frequents(x,z))

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

Q(x) = ∃y. Likes(x, y)∧∀z.(Serves(z,y) ⇒ Frequents(x,z))

≡ Q(x) = ∃y. Likes(x, y)∧∀z.(¬ Serves(z,y) ∨ Frequents(x,z))

Step 1: Replace ∀ with ∃ using de Morgan's Laws

∀x P(x) same as ¬∃x ¬P(x)

Q(x) = ∃y. Likes(x, y)∧ ¬∃z.(Serves(z,y) ∧ ¬Frequents(x,z))

¬(¬P∨Q) same as P∧ ¬ Q

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

From RC to SQL

Query: Find drinkers that like some beer so much that they frequent all bars that serve it

Q(x) = ∃y. Likes(x, y) ∧¬ ∃z.(Serves(z,y)∧¬Frequents(x,z))

Step 2: Translate into SQL

```
SELECT DISTINCT L.drinker
FROM Likes L
WHERE not exists
  (SELECT S.bar
   FROM Serves S
   WHERE L.beer=S.beer
      AND not exists (SELECT *
                      FROM Frequents F
                      WHERE F.drinker=L.drinker
                         AND F.bar=S.bar))
```

We will see a "methodical and correct" translation trough "safe queries" in Datalog

Summary

- You learnt three query languages for the Relational DB model
  - SQL
  - RA
  - RC
- All have their own purposes
- You should be able to write a query in all three languages and convert from one to another
  - However, you have to be careful, not all "valid" expressions in one may be expressed in another
  - {S | ¬ (S ∈ Sailors)} – infinitely many tuples – an "unsafe" query
  - More when we do "Datalog", also see Ch. 4.4 in [RG]

Where are we now?

We learnt

- ✓Relational Model and Query Languages
  - ✓SQL, RA, RC
  - ✓Postgres (DBMS)
  - ✓XML (overview)
  - HW1

Next

- Database Normalization
  - (for good schema design)

Design Theory and Normalization

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.
- Some slides have been adapted from slides by Prof. Jun Yang

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

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

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?

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?

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 a possible FD here?

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
  - denoting 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}
  - here first letter of each attribute
- 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 ⊇ Y, then X → Y
- Augmentation: If X → Y, then XZ → YZ for any Z
- Transitivity: If X → Y and Y → Z, then X → 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 → C and check

Armstrong's Axioms

- X, Y, Z are sets of attributes
- Reflexivity: If X ⊇ Y, then X → Y
- 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
  - 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 → Y and X → Z, then X → YZ
- Decomposition: If X → YZ, then X → Y and X → Z

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

A → B, A → C
A → BC

A → BC
A → B, A → 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^+

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

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
- 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

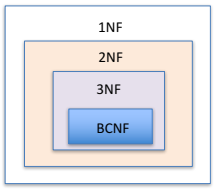
FDs play a role in detecting redundancy

Example

- Consider a relation R with 3 attributes, ABC
  - No FDs hold: There is no redundancy here – no decomposition needed
  - Given $A \rightarrow B$: 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
- Intuitive idea:
  - if there is any non-key dependency, e.g. $A \rightarrow B$, decompose!

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 $X \rightarrow A$ in F
  - $A \in X$ (called a trivial FD), or
  - X contains a key for R
    - i.e. X is a superkey

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

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?

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

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?

Lossless decomposition of R into R1, R2 happens if
- either $R1 \cap R2 \rightarrow R1$
- or $R1 \cap R2 \rightarrow R2$

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

- NOTE: Need to consider all possible FDs that can be inferred from the current set of FDs (closure), not only the given ones!
- Gives a collection of relations that are
  - in BCNF
  - lossless join decomposition
  - and guaranteed to terminate

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

BCNF decomposition example

uid → uname, twitterid
twitterid → uid
uid, gid → fromDate

UserJoinsGroup (*uid*, *uname*, *twitterid*, *gid*, *fromDate*)

Ack: Slide from Prof. Jun Yang

Another example

uid → uname, twitterid
twitterid → uid
uid, gid → fromDate

UserJoinsGroup (*uid*, *uname*, *twitterid*, *gid*, *fromDate*)

Ack: Slide from Prof. Jun Yang

43

Recap

- Functional dependencies: a generalization of the key concept
- Non-key functional dependencies: a source of redundancy
- BCNF decomposition: a method for removing redundancies
  - BNCF decomposition is a lossless join decomposition
- BCNF: schema in this normal form has no redundancy due to FD's

44

BCNF = no redundancy?

- *User* (*uid*, *gid*, *place*)
  - A user can belong to multiple groups
  - A user can register places she's visited
  - Groups and places have nothing to do with other
  - FD's?
  - BCNF?
  - Redundancies?

uid	*gid*	*place*
142	dps	Springfield
142	dps	Australia
456	abc	Springfield
456	abc	Morocco
456	gov	Springfield
456	gov	Morocco
…	…	…

45

Multivalued dependencies

- A multivalued dependency (MVD) has the form $X \twoheadrightarrow Y$, where X and Y are sets of attributes in a relation R
- $X \twoheadrightarrow Y$ means that whenever two rows in R agree on all the attributes of X, then we can swap their Y components and get two rows that are also in R

X	Y	Z
a	b_1	c_1
a	b_2	c_2
a	b_2	c_1
a	b_1	c_2
…	…	…

46

MVD examples

User (*uid*, *gid*, *place*)

- *uid* ↠ *gid*
- *uid* ↠ *place*
  - Intuition: given *uid*, attributes *gid* and *place* are "independent"
- *uid*, *gid* ↠ *place*
  - Trivial: LHS ∪ RHS = all attributes of R
- *uid*, *gid* ↠ *uid*
  - Trivial: LHS ⊇ RHS

47

Verify these yourself!

Complete MVD + FD rules

- FD reflexivity, augmentation, and transitivity
- MVD complementation:
  If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow attrs(R) - X - Y$
- MVD augmentation:
  If $X \twoheadrightarrow Y$ and $V \subseteq W$, then $XW \twoheadrightarrow YV$
- MVD transitivity:
  If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z - Y$
- Replication (FD is MVD):
  If $X \rightarrow Y$, then $X \twoheadrightarrow Y$
- Coalescence:
  If $X \twoheadrightarrow Y$ and $Z \subseteq Y$ and there is some W disjoint from Y such that $W \rightarrow Z$, then $X \rightarrow Z$

Try proving things using these!?

48

An elegant solution: "chase"

- Given a set of FD's and MVD's $\mathcal{D}$, does another dependency d (FD or MVD) follow from $\mathcal{D}$?
- Procedure
  - Start with the premise of d, and treat them as "seed" tuples in a relation
  - Apply the given dependencies in $\mathcal{D}$ repeatedly
    - If we apply an FD, we infer equality of two symbols
    - If we apply an MVD, we infer more tuples
  - If we infer the conclusion of d, we have a proof
  - Otherwise, if nothing more can be inferred, we have a counterexample

49

Proof by chase

- In $R(A,B,C,D)$, does $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$ imply that $A \twoheadrightarrow C$?

Have:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Need:

A	B	C	D
a	b_1	c_2	d_1
a	b_2	c_1	d_2

$A \twoheadrightarrow B$

a	b_2	c_1	d_1
a	b_1	c_2	d_2

$B \twoheadrightarrow C$

a	b_2	c_1	d_2
a	b_2	c_2	d_1

$B \twoheadrightarrow C$

a	b_1	c_2	d_1
a	b_1	c_1	d_2

50

Another proof by chase

- In $R(A,B,C,D)$, does $A \to B$ and $B \to C$ imply that $A \to C$?

Have:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Need: $c_1 = c_2$

$A \to B$ $\quad b_1 = b_2$

$B \to C$ $\quad c_1 = c_2$

In general, with both MVD's and FD's, chase can generate both new tuples and new equalities

51

Counterexample by chase

- In $R(A,B,C,D)$, does $A \twoheadrightarrow BC$ and $CD \to B$ imply that $A \to B$?

Have:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Need: $b_1 = b_2$

$A \twoheadrightarrow BC$

a	b_2	c_2	d_1
a	b_1	c_1	d_2

Counterexample!

52

4NF

- A relation R is in Fourth Normal Form (4NF) if
  - For every non-trivial MVD $X \twoheadrightarrow Y$ in R, X is a superkey
  - That is, all FD's and MVD's follow from "key → other attributes" (i.e., no MVD's and no FD's besides key functional dependencies)
- 4NF is stronger than BCNF
  - Because every FD is also a MVD

53

4NF decomposition algorithm

- Find a 4NF violation
  - A non-trivial MVD $X \twoheadrightarrow Y$ in R where X is not a superkey
- Decompose R into R_1 and R_2, where
  - R_1 has attributes $X \cup Y$
  - R_2 has attributes $X \cup Z$ (where Z contains R attributes not in X or Y)
- Repeat until all relations are in 4NF
- Almost identical to BCNF decomposition algorithm
- Any decomposition on a 4NF violation is lossless

54

4NF decomposition example

uid	gid	place
142	dps	Springfield
142	dps	Australia
456	abc	Springfield
456	abc	Morocco
456	gov	Springfield
456	gov	Morocco
…	…	…

User (*uid*, *gid*, *place*)
4NF violation: *uid* ↠ *gid*

Member (*uid*, *gid*)
4NF

uid	gid
142	dps
456	abc
456	gov
…	…

Visited (*uid*, *place*)
4NF

uid	place
142	Springfield
142	Australia
456	Springfield
456	Morocco
…	…

55

Other kinds of dependencies and normal forms

- Dependency preserving decompositions
- Join dependencies
- Inclusion dependencies
- 5NF
- See book if interested (not covered in class)

Summary

- Philosophy behind BCNF, 4NF:
  Data should depend on the key, the whole key, and nothing but the key!
  - You could have multiple keys though
- Redundancy is not desired typically
  - not always, mainly due to performance reasons
- Functional/multivalued dependencies – capture redundancy
- Decompositions – eliminate dependencies
- Normal forms
  - Guarantees certain non-redundancy
  - 3NF, BCNF, and 4NF
- Lossless join
- How to decompose into BCNF, 4NF
- Chase

57