Functional Dependencies and Schema Refinement

Review: Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level descr (often done w/ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - consistency,normalization
- Physical Design
  - indexes, disk layout
- Security Design
  - who accesses what

Review: Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level descr (often done w/ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - consistency,normalization
- Physical Design
  - indexes, disk layout
- Security Design
  - who accesses what

The Evils of Redundancy

- Redundancy in relational schemas
  - leads to wasted storage
  - more important: insert/delete/update anomalies
- Solution: *Functional Dependencies*
  - a form of integrity constraints
  - help identify redundancy in schemas
  - help suggest *refinements*
- Main refinement technique: *Decomposition*
  - split the columns of one table into two tables
  - often good, but need to do this judiciously

Functional Dependencies (FDs)

- Idea: $X \rightarrow Y$ means
  - Given any two tuples in table r,
    if their X values are the same,
    then their Y values must be the same.
    (but not vice versa)
  - (Read "→" as "*determines*")
- Formally: An FD $X \rightarrow Y$ holds over relation schema R if, for *every allowable instance* r of R:

$$t1 \in r,\ \ t2 \in r,\ \ \pi_X(t1) = \pi_X(t2)$$

$$\textit{implies}\quad \pi_Y(t1) = \pi_Y(t2)$$

($t1$, $t2$ are tuples; X, Y are *sets* of attributes)

FD's Continued

- An FD is w.r.t. *all* allowable instances.
  - *Declared* based on app semantics
  - *Not* learned from data
    - (though you might learn *suggestions* for FDs)
- Question: How related to keys?
  - "K → [all attributes of R]"? K is a *superkey* for R! (does not require K to be *minimal*.)
  - FDs are a generalization of keys.

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:
  Hourly_Emps (ssn, name, lot, rating, wage_per_hr, hrs_per_wk)
- We can denote a relation schema by listing its attributes:
  - e.g., SNLRWH
  - This is really the set of attributes {S,N,L,R,W,H}.
- And we can use relation name to refer to the set of all its attributes
  - e.g., "Hourly_Emps" for SNLRWH
- What are some FDs on Hourly_Emps?

  ssn is the primary key: S → SNLRWH

  rating determines *wage_per_hr*: R → W

  lot determines *lot*: L → L ("trivial" dependency)

Problems Due to R → W

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

Hourly_Emps

- *Update anomaly*: Can we modify W in only the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his or her rating? (or we get it wrong?)
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Detecting Reduncancy

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

Hourly_Emps

Q: Why was R → W problematic, but S→W not?

Decomposing a Relation

- Redundancy can be removed by "chopping" the relation into pieces.
- FD's are used to drive this process.
  - R → W is causing the problems, so decompose SNLRWH into what relations?

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

Hourly_Emps2

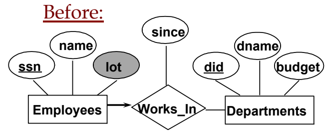
R	W
8	10
5	7

Wages

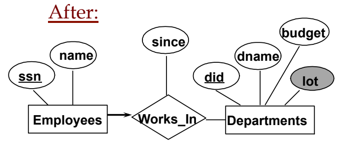
Refining an ER Diagram

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

Before:



After:



Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:

  title → *studio, star* implies *title* → *studio* and *title* → *star*

  title → *studio* and *title* → *star* implies *title* → *studio, star*

  title → *studio* and *studio* → *star* implies *title* → *star*

  But,
  title, star → *studio* does **NOT** necessarily imply that *title* → *studio* or that *star* → *studio*

- An FD *g* is *implied by* a set of FDs *F* if *g* holds whenever all FDs in *F* hold.
- F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")

Rules of Inference

- 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 $X \rightarrow Z$
- S*ound* and *complete* inference rules for FDs!
  - using AA you get *only* the FDs in F+ and *all* these FDs.
- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Example

- Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - C is the key: $C \rightarrow CSJDPQV$
  - Proj (J) purchases each part (P) using single contract (C): $JP \rightarrow C$
  - Dept (D) purchases at most 1 part (P) from a supplier (S): $SD \rightarrow P$
- Problem: Prove that SDJ is a key for Contracts
- $JP \rightarrow C$, $C \rightarrow CSJDPQV$ imply $JP \rightarrow CSJDPQV$
  (by transitivity) (shows that JP is a key)
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$
  (by augmentation)
- $SDJ \rightarrow JP$, $JP \rightarrow CSJDPQV$ imply $SDJ \rightarrow CSJDPQV$
  (by transitivity) (shows that SDJ is a key).

Q: can you now infer that $SD \rightarrow CSDPQV$
(i.e., drop J on both sides)? **No!**

Attribute Closure

- Computing closure F^+ of a set of FDs F is hard:
  - exponential in # attrs!
- Typically, just check if $X \rightarrow Y$ is in F^+. Efficient!
  - Compute *attribute closure* of X (denoted X^+) wrt F.
    X^+ = Set of all attributes A such that $X \rightarrow A$ is in F^+
    - $X^+ := X$
    - Repeat until no change (fixpoint):
      if $U \rightarrow V \subseteq F$, $U \subseteq X^+$, then add V to X^+
  - Check if Y is in X^+
  - Approach can also be used to find the keys of a relation.
    - If $X^+ = R$, then X is a superkey for R.
    - Q: How to check if X is a "candidate key" (minimal)?

Attribute Closure (example)

- R = {A, B, C, D, E}
- F = { $B \rightarrow CD$, $D \rightarrow E$, $B \rightarrow A$, $E \rightarrow C$, $AD \rightarrow B$ }

- **Is $B \rightarrow E$ in F^+ ?**
  B+ = ...
  ... Yep!
- **Is D a key for R?**
  D^+ = ...
  ... Nope!
- **Is AD a key for R?**
  AD^+ = ...
  ...Yep!
- **Is AD a *candidate* key for R?**
  A^+ = ... D^+ = ...
  ...Yes!
- **Is ADE a *candidate* key for R?**
  No!

Thanks for that…

- So we know a lot about FDs
- So what?
- Can they help with schema refinement?

Normal Forms

- Q1: is any refinement is needed??!
- If relation is in a *normal form* (BCNF, 3NF etc.):
  - we know certain problems are avoided/minimized.
  - helps decide whether decomposing relation is useful.
- Consider a relation R with 3 attributes, ABC.
  - No (non-trivial) FDs hold: No redundancy here.
  - Given $A \rightarrow B$: If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!

Basic Normal Forms

- 1st Normal Form – all attributes atomic
  - I.e. relational model
  - Violated by many common data models
    - Including XML, JSON, various OO models
  - Some of these "non-first-normal form" (NFNF) quite useful in various settings
    - especially in update-never, fixed-query settings
    - if you never "unnest", then who cares!
      - basically relational collection of structured objects
- 1st ⊃ 2nd (of historical interest) ⊃ 3rd ⊃ Boyce-Codd ⊃ …

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for all $X \rightarrow A$ in F+
  - $A \subseteq X$ (called a trivial FD), or
  - X is a superkey for R.
- In other words: "R is in BCNF if the only non-trivial FDs over R are key constraints."
- If R in BCNF, every field of every tuple records useful info that cannot be inferred via FDs alone.
  - Say we know FD $X \rightarrow A$ holds for this example relation:
  - Can you guess the value of the missing attribute
  - Yes, so relation is not in BCNF
- Connection here to compression/information theory

X	Y	A
x	y1	a
x	y2	?

Decomposition of a Relation Scheme

- How to normalize a relation?
  - *decompose* into multiple normalized relations
- Suppose *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
  - Every attribute of R appears as an attribute of at least one of the new relations.

Example (same as before)

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

Hourly_Emps

- SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
- Q: Is this relation in BCNF?

No, The second FD causes a violation;
W values repeatedly associated with R values.

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

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

Hourly_Emps2

R	W
8	10
5	7

Wages

- Q: Are both of these relations are now in BCNF?
- **Decompositions should be used only when needed.**
  - Q: potential problems of decomposition?

Problems with Decompositions

- There are three potential problems to consider:
  1) May be ***impossible*** to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

Tradeoff: **Must consider these 3 vs. redundancy.**

Lossless Decomposition (example)

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

$\bowtie$

R	W
8	10
5	7

=

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

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	B
1	2
4	5
7	2

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Join Decompositions

- Defn: 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 $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold!
  - If it does, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem #1)*

More on Lossless Decomposition

- The decomposition of R into X and Y is lossless with respect to F *if and only if* the closure of F contains:

  $X \cap Y \rightarrow X$, **or**

  $X \cap Y \rightarrow Y$

  in example: decomposing ABC into AB and BC is lossy, because intersection (i.e., "B") is not a key of either resulting relation.

- Useful result: If $W \rightarrow Z$ holds over R and $W \cap Z$ is empty, then decomposition of R into R-Z and WZ is loss-less.

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	C
1	3
4	6
7	8

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Dependency Preserving Decomposition

- Dependency preserving decomposition (Intuitive):
  - A decomposition where the following is true:

    If R is decomposed into X, Y and Z,
    and we enforce FDs individually on each of X, Y and Z,
    then all FDs that on R must also hold on result.
    (Avoids Problem #2 on our list.)

- *Defn: Projection of set of FDs F* :
  If R is decomposed into X and Y
  the projection of F on X (denoted F_X)
  is the set of FDs $U \rightarrow V$ in F^+
  such that all of the attributes U, V are in X.

F+: closure of F , not just F!

Dependency Preserving Decompositions (Contd.)

- Defn: Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
  - (just the formalism of our intuition above)
- Important to consider F^+ in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?????
- Note: F^+ contains $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$, so…
  - $F_{AB} \supseteq \{A \rightarrow B, B \rightarrow A\}$; $F_{BC} \supseteq \{B \rightarrow C, C \rightarrow B\}$
  - So, $(F_{AB} \cup F_{BC})^+ \supseteq \{C \rightarrow A\}$

Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow Y$ violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
- e.g., CSJDPQV, key C, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - To deal with $SD \rightarrow P$, decompose into SDP, CSJDQV.
  - To deal with $J \rightarrow S$, decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV
- Note: several dependencies may cause violation of BCNF. The order in which we "deal with" them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, $CS \rightarrow Z$, $Z \rightarrow C$
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs $JP \rightarrow C$, $SD \rightarrow P$ and $J \rightarrow S$).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - but JPC tuples are stored only for checking the f.d. (*Redundancy!*)

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is *prime*")
- *Minimality* of a candidate 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'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

3NF vs. BCNF

- Reln R with FDs F is in BCNF if, for all $X \rightarrow A$ in F+
  - $A \subseteq X$ (called a trivial FD), or
  - X is a superkey for R.
- Reln R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is *prime*")

Third Normal Form (3NF)

- 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'' decomp, or performance considerations).
- *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- If 3NF violated by X → A, we can look at only two possible cases:
  - X is a subset of some key K ("partial dependency")
    - We store (X, A) pairs redundantly.
    - e.g. Reserves SBDC (C is for credit card) with key SBD and S → C
  - X is not a proper subset of any key. ("transitive dep.")
    - 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 (different K's, same X implies same A!) – problem with initial SNLRWH example.
- But: even if R is in 3NF, these problems could arise.
  - e.g., Reserves SBDC
    S → C, C → S is in 3NF (why?),
    but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.
- To ensure dependency preservation, one idea:
  - If X → Y is not preserved, add relation XY.

  Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' JP → C. What if we also have J → C ?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F.*

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- e.g., A → B, ABCD → E, EF → GH, ACDF → EG has the following minimal cover:
  - A → B, ACD → E, EF → G and EF → H
- M.C. implies Lossless-Join, Dep. Pres. Decomp!!!
  - (in book)

Summary of Schema Refinement

- BCNF: each field contains information that cannot be inferred using only FDs.
  - ensuring BCNF is a good heuristic.
- Not in BCNF? Try decomposing into BCNF relations.
  - Must consider whether all FDs are preserved!
- Lossless-join, dependency preserving decomposition into BCNF impossible? Consider 3NF.
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.
- Note: even more restrictive Normal Forms exist (we don't cover them in this course, but some are in the book.)