Schema Refinement and Normal Forms

CS430/630
Lecture 16

Slides based on "Database Management Systems" 3rd ed, Ramakrishnan and Gehrke

Why Schema Refinement?

- We have learnt the advantages of relational tables …
- … but how to decide on the relational schema?

- At one extreme, store everything in single table
  - Huge redundancy
  - Leads to anomalies!

- We need to break the information into several tables
  - How many tables, and with what structures?
  - Having too many tables can also cause problems
    - E.g., performance, difficulty in checking constraints

Sample Relation

Hourly_Emps (*ssn*, *name*, *lot*, *rating*, *wage*, *hrs_worked*)

- Denote relation schema by attribute initial: SNLRWH
- Constraints (dependencies)
  - *ssn* is the key: S → SNLRWH
  - *rating* determines *wage*: R → W
    - E.g., worker with rating *A* receives 20$/hr

Anomalies

- Problems due to R ⟶ W :
  - *Update anomaly*: Change value of *W* only in a tuple – dependency violation
  - *Insertion anomaly*: How to insert employee if we don't know hourly wage for that rating?
  - *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

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

Removing Anomalies

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

Wages

R	W
8	10
5	7

Create 2 smaller tables!

- Updating rating of employee will result in the wage "changing" accordingly
  - Note that there is no physical change of W, just a "pointer change"
- Deleting employee does not affect rating-wages data

Dealing with Redundancy

- *Redundancy* is at the root of redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify redundancy
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD)
- Decomposition should be used judiciously:
    - Decomposition may sometimes affect performance. Why?
    - What problems (if any) does decomposition cause?
        - Incorrect data
        - Loss of dependencies

Functional Dependencies (FDs)

- A functional dependency X → Y holds over relation R if for every instance r of R
  - $t1, t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - given two tuples in *r*, if the X values agree, Y values must also agree

- FD is a statement about *all* allowable relations.
  - Identified based on semantics of application (business logic)
  - Given an instance *r* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!

FDs and Keys

- FDs are a generalization of keys
  - A key uniquely identifies all attribute values in a tuple
  - That is a particular case of FD …
  - … but not all FDs must determine ALL attributes

- K is a key for R means that K → R
  - However, K → R does not require K to be *minimal*!
  - K can be a superkey as well

Reasoning About FDs

- Given FD set *F*, we can usually infer additional FDs:
  - 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 Y $\subseteq$ X, 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!

Reasoning About FDs (cont'd)

- Additional rules
  - Not necessary, but helpful

- Union and decomposition (splitting)
  - X →Y and X →Z => X →YZ
  - X →YZ => X →Y and X →Z

An Example of FD Inference

- Contracts(*cid, sid, jid, did, pid, qty, value*), and:
  - Contract id, supplier, project, department, part
  - C is the key: C → CSJDPQV
  - Project purchases each part using single contract: JP → C
  - Dept purchases at most one part from a supplier: SD → P

- JP → C, C → CSJDPQV imply JP → CSJDPQV
- SD → P implies SDJ → JP
- SDJ → JP, JP → CSJDPQV imply SDJ → CSJDPQV

Attribute Closure

- *Attribute closure* of X (denoted X^+) wrt FD set F:
  - Set of all attributes A such that X → A is in F^+
  - Set of all attributes that can be determined starting from attributes in X and using FDs in F

- Apply split rule such that all FDs have single attr in RHS

```
$X^+ = X$
Repeat
    $Y = X^+$
    Search all FDs in F with LHS completely included in $X^+$
    Add RHS of those FDs to $X^+$
Until $Y = X^+$
```

Verifying if given FD in FD-set closure

- Computing the closure of a set of FDs can be expensive
  - Size of closure is exponential in number of attributes!

- But if we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F:
  - Can be done efficiently without need to know F^+
  - Compute X^+ wrt F
  - Check if Y is in X^+

Verifying if attribute set is a key

- Key verification can also be done with attribute closure

- To verify if X is a key, two conditions needed:
  - $X^+ = R$
  - X is minimal

- How to test minimality
  - Removing an attribute from X results in X' such that $X'^+ <> R$