Database Management Systems
L8

Umass Boston
Summer 2023
Cristina Maier

Slides are based on “Database Management System, 3rd ed., by Ramakrishnan and Gehrke

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design: the Entity-Relationship Model
- Structured Query Language (SQL)
- Database Security and Authorization
- Schema Refinement and Normal Forms
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

Schema Refinment and Normal Forms

- We have learnt the advantages of relational tables
- 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

Schema Relation

Hourly_Emps (<u>*ssn*</u>, *name*, *lot*, *rating*, *wage*, *hrs_worked*)

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

Anomalies

- Problems due to R $\rightarrow$ 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

- Creating two 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

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

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

- A <u>functional dependency</u> 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 tables in r, if the X value 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 $X \rightarrow Y$
  - *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!

Reasoning about FDs (cont.)

- Additional rules
  - Not necessary, but helpful
- Union and decomposition (splitting)
  - $X \rightarrow Y$ and $X \rightarrow Z \Rightarrow X \rightarrow YZ$
  - $X \rightarrow YZ \Rightarrow X \rightarrow Y$ and $X \rightarrow Z$

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 $\rightarrow$ CSJDPQV
  - Project purchases each part using single contract:
  - JP $\rightarrow$ C
  - Dept purchases at most one part from a supplier:
  - SD $\rightarrow$ P

- JP $\rightarrow$ C, C $\rightarrow$ CSJDPQV imply JP $\rightarrow$ CSJDPQV
- SD $\rightarrow$ P implies SDJ $\rightarrow$ JP
- SDJ $\rightarrow$ JP, JP $\rightarrow$ CSJDPQV imply SDJ $\rightarrow$ CSJDPQV

Attribute Closure

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

Verifying if given FD is in given FD-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 $X'^+ <> R$
-

Reasoning about FDs (recap.)

- 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 $X \rightarrow Y$
  - *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!

Reasoning about FDs (recap.)

- Additional rules
  - Not necessary, but helpful
- Union and decomposition (splitting)
  - $X \rightarrow Y$ and $X \rightarrow Z \Rightarrow X \rightarrow YZ$
  - $X \rightarrow YZ \Rightarrow X \rightarrow Y$ and $X \rightarrow Z$

Normal Forms

- Decompositions:
  - BCNF
  - 3NF

Decomposition of a Relational Schema

- A *decomposition* of R replaces it by two or more relations
  - Each new relation schema contains a subset of the attributes of R
  - Every attribute of R appears in one of the new relations
  - E.g., SNLRWH decomposed into SNLRH and RW
- Decompositions should be used only when needed
  - Cost of join will be incurred at query time
- Problems may arise with (improper) decompositions
  - Reconstruction of initial relation may not be possible
  - Dependencies cannot be checked on smaller tables

Lossless Join Decompositions

- Decomposition of R into X and Y is *lossless-join* if:

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

- 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.
-

- *It is essential that all decompositions used to deal with redundancy be lossless!*

Incorrect Decomposition

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

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

Natural Join

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

Condition for Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$
- In particular, the decomposition of R into UV and R - V is lossless-join if $U \rightarrow V$ holds over R.

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - Consider decomposition: CSJDQV and SDP
  - Problem: Checking JP → C requires a join!
- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X and Y, and we enforce the FDs that hold on X, Y then all FDs that were given to hold on R must also hold
- *Projection of set of FDs F*: If R is decomposed into X, ... projection of F onto X (denoted F_X) is the set of FDs U → V in F^+ (*closure of F*) such that U, V are in X.

Dependency Preserving Decomposition

- Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$
  - Dependencies that can be checked in X without considering Y, and in Y without considering X, together represent all dependencies in F^+

- Dependency preserving does not imply lossless join:
- ABC, $A \rightarrow B$, decomposed into AB and BC.
-

Two kind of possible problems with Decomposition

- Not being - Lossless join
- Not Preserving Dependency

Normal Forms

- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized.
- Role of FDs in detecting redundancy:
  - Consider a relation R with attributes AB
    - No FDs hold: There is no redundancy
    - Given A $\rightarrow$ B:
      - Several tuples could have the same A value
      - If so, they'll all have the same B value!

Boyce-Codd Normal Form (BCNF)

- Relation R with FDs F is in BCNF if, for all $X \rightarrow A$ in F^+
  - $A \subseteq X$ (called a *trivial* FD), or
  - X contains a key for R
- The only non-trivial FDs allowed are key constraints
- BCNF guarantees no anomalies occur

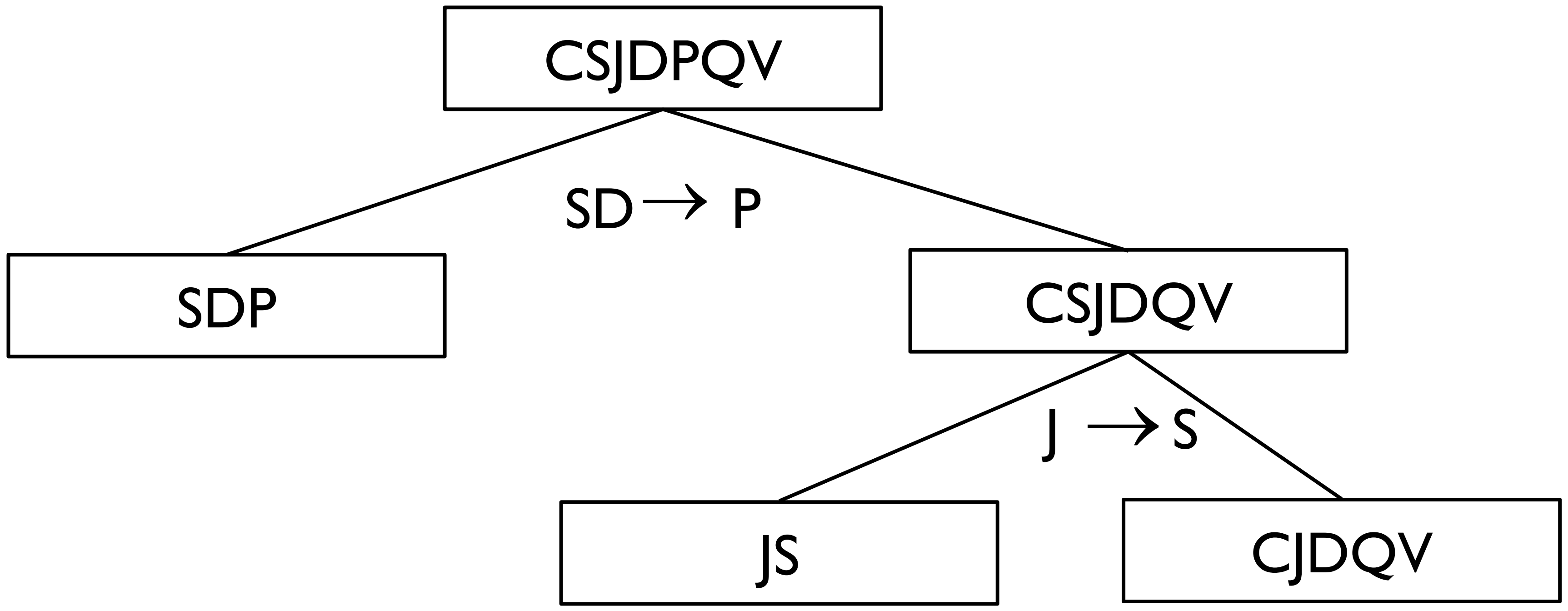
Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R - Y and XY.

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, SD → P, J → S
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV

Decomposition into BCNF



- In general, several dependencies may cause violation of BCNF. The order in which we “deal with” them could lead to very different sets of relations!

BCNF Decomposition

- Guarantees lossless join
- Does not always guarantee dependency preservation

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF
  - e.g., $\underline{AB}C$, $AB \rightarrow C$, $C \rightarrow A$
    - Can't decompose while preserving first FD; in this case it cannot preserve dependency!
    - CA , AB
-

Third Normal form (3NF)

- Relation R with FDs F is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \subseteq X$ (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R (A here is a single attribute)
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, it is also in 3NF.
- If R is in 3NF, some redundancy is possible
  - compromise used when BCNF not achievable
  - e.g., no ``good'' decomposition, or performance considerations
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

Decomposition into 3NF

- Lossless join decomposition algorithm also applies to 3NF
- To ensure dependency preservation, one idea:
  - If X→Y is not preserved, add relation XY
  - Refinement: Instead of the given set of FDs F, use a *minimal cover for F*
  - Example: CSJDPQV, JP → C, SD → P, J → S
  - Choose SD → P, result is SDP and CSJDQV
  - Choose J → S, result is JS and CJDQV, all 3NF
  - Add CJP relation

Summary of Schema Refinement

- BCNF: relation is free of FD redundancies
  - Having only BCNF relations is desirable
  - If relation is not in BCNF, it can be decomposed to BCNF
    - Lossless join property guaranteed
    - But some FD may be lost
- 3NF is a relaxation of BCNF
  - Guarantees both lossless join and FD preservation
- Decompositions may lead to performance loss
  - *performance requirements* must be considered when using decomposition

Two kind of possible problems with Decomposition (recap.)

- Not being - Lossless join
- Not Preserving Dependency

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 first FD;
- Similarly, decomposition of CSJDQV 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.
    - JPC tuples stored only for checking FD! (*Redundancy!*)

Schema Relation

Hourly_Emps (<u>*ssn*</u>, *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

Decomposing

- S is key; R —> W
- SNLRWH into RW , SNLRH. This is in BCNF

Questions?