AY2023/24 Trimester 3

Dr. ZHANG Wei

INF2003: Database Systems

Database Refinement

2

SIT Internal

Database Refinement

- Given an application, the database design is **not unique**.
- With many choices, which is the best one?
  - We might never know.
  - But very often, our design is not the best. Space to improve.
- How to **define** better or **best**?
  - No **missing** information.
  - No unnecessary **redundancy**.
    - Some redundancy cannot be avoided.
  - Easy for information **management**.
  - Easy to add **constraints**.

ssn	name	lot	rating	wage	hours
1	Alice	48	8	10	40
2	Bob	22	8	10	30
3	Catelyn	35	5	7	30
4	Dolores	35	5	7	32
5	Evan	35	8	10	40

- **Redundancy** (store the same information in ≥ 1 places in the database).
  - Requirement: 1 rating value <-> 1 wage.
  - **Redundant storage**, e.g., repeated value 8.
  - **Update anomalies**, e.g., update wage 10 -> update 1 row? all rows?
  - **Insertion anomalies**, e.g., insert a new rating with unknown wages, doable?
  - **Deletion anomalies**, e.g., if we delete rating 5, we lose the mapping.

3

SIT Internal

Decomposition

- Redundancy: forces **unnatural association** between the attributes.
- **Functional dependency** (FD): can identify such situations and suggest refinement to the schema.
- Idea: replace a relation with a collection of **'smaller' relations.**
- **Decomposition**: replacing relation schema by ≥ 2 schemas each with a **subset of the attributes** and together include all attributes.
- Example: decompose HourlyEmps into 2 relations.
  - Specific rating-wage relation, even no employee for some ratings.
  - Change any wage? Just update the wages tuple.
  - Efficient. Eliminate potential redundancy.

ssn	name	lot	rating	wage	hours
1	Alice	48	8	10	40
2	Bob	22	8	10	30
3	Catelyn	35	5	7	30
4	Dolores	35	5	7	32
5	Evan	35	8	10	40

ssn	name	lot	rating	hours
1	Alice	48	8	40
2	Bob	22	8	30
3	Catelyn	35	5	30
4	Dolores	35	5	32
5	Evan	35	8	40

rating	wage
8	10
5	7

4

SIT Internal

Normal Forms

- **Refinement/decomposition: when do we need to** perform?
- Fact: if a relation is in a certain **normal form** (e.g., BCNF, 3NF, etc.) -> certain kinds of problems can be avoided/minimized.
  - Choose a suitable normal form.
- Idea: **replace** the current relational schema with a new (better) one.
- **1NF**: All rows must contain the same number of fields (e.g., no lists, no repeated attributes).
- 2NF: old and obsolete.
- 3NF: defined by E. F. Codd in 1971.
- **BCNF**: developed in 1975 by Raymond F. Boyce and Edgar F. Codd.
- 4NF and 5NF (not covered and not popular).

5

Functional Dependency (FD) for Decomposition

- **FD**: kind of integrity constraint that **generalizes** the concept of **key**.
- Definition: If two tuples agree on the attributes A1, A2, …, An, then they must also agree on the attributes B1, B2, …, Bm.
- Formally: A1, A2, …, An → B1, B2, …, Bm
- X → Y is read as : X functionally **determines** Y or X determines Y.
- An FD **holds**, or **does not hold**?
  - EmpID → Name, Phone, Position
  - Position → Phone
  - Phone → Position

id	A1	A2	…	An		B1	B2	…	Bm
t					→				
t’					→				

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

…	Phone	Position
…	1234	Clerk
…	9876 ←	Salesrep
…	9876 ←	Salesrep
…	1234	Lawyer

EmpID	Name	Phone	Position
E0045	Smith	1234	→ Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	→ Lawyer

6

FD Example

- A **legal** instance must satisfy ALL specified ICs, including FDs.
  - FD is a statement about all possible legal instances of the relation.
  - Looking at an instance, we can tell that a certain FD does not hold.
  - We can never deduce that an FD does hold just by looking at one instance.
- FD: AB -> C. (The instance satisfies?)
  - (See the 1st 2 tuples) An FD is not the same as a key constraint.
  - (See the 3rd and 4th tuples) If two tuples differ in either the field A or B, they must differ in the C field without violating the FD.
  - Can we add <a1, b1, c2, d1>?
- Reality: **do not know all** FDs initially.
- How to find all?

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

7

SIT Internal

FD – Inference Rule

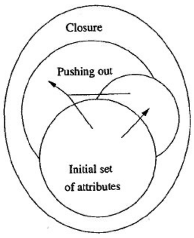
- Given a set of FDs over relation schema R, we can **infer** additional FDs.
  - Workers(ssn, name, lot, did, since)
  - ssn -> did, did -> lot implies ssn -> lot.
- Requirement: an FD f is **implied** by a given set F of FDs if f holds whenever **all** FDs in F hold.
- **Armstrong's Axioms**.
  - Let X, Y, Z denote sets of attributes over relation R.
- **Reflexivity**: if X ⊆ Y, then Y -> X.
- **Augmentation**: if X -> Y, then XZ -> YZ for any Z.
- **Transitivity**: if X -> Y and Y -> Z, then X -> Z.
- Find all FDs? The 3 simple rules are enough.
- However, clumsy to use in practice and expensive.

8

SIT Internal

Inference Rules and Closure Algorithm

- Better: use **closure** of a set of attributes.
  - **Closure** of F, denoted as **F+**:
- Algorithm:
  - Input: attributes {A1, …, An} and set of FD S.
  - Output: {A1, …, An}+
  - Split FDs of S each FD has single attribute on right.
  - X <- {A1, …, An} (temporary closure).
  - **Repeat** until X does not change.
    - if B1, …, Bn -> C is a FD and B1, …, Bn are all in X.
    - then add C to X.



9

SIT Internal

Computing Closure – Example

name -> color
category -> department
color, category -> price

- Question: what is {name, category}+?
- Input: {name, category} and FDs.
- Split FD with single attribute on RHS.
- Initialize X = {name, category}.
  - Look at FD **name -> color** AND name ∈ X, so add color to X.
    - X = {name, category, color}
  - Look at FD **category -> department** AND category ∈ X, so add department to X.
    - X = {name, category, color, department}
  - Look at FD **color, category -> price** AND {color, category} are all in X, so add price to X.
    - X = {name, category, color, department, price}
- X = {name, category, color, department, price}. (All)
- So {name, category}+ = {name, category, color, department, price}.
- Or, name, category -> color, department, price.

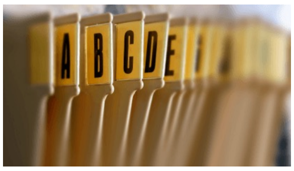
10

SIT Internal

Eliminating Anomalies

- **Goal** of decomposition: replace a relation by several **without anomalies**.
- **BCNF**: a condition under which we can guarantee there is no anomalies.
- Definition: if and only if for every one of its dependencies X -> Y:
  - X -> Y is a **trivial** FD (or Y is a subset of X).
  - **Or** X is a **super-key** for R.
- Equivalently: ∀X, either (X+ = X) OR (X+ = all attributes)
- Overall, relation in BCNF will have no "bad" FDs.
- Example: Product(name, price, category, color).
  - (name, category)+ = name, category, price, color.
  - It is a key. Good FD.
  - (category) + = category, color.
  - So, (category) is NOT a key. **Bad FD**.
  - Relation Product is **NOT in BCNF**.

name, category -> price
category -> color

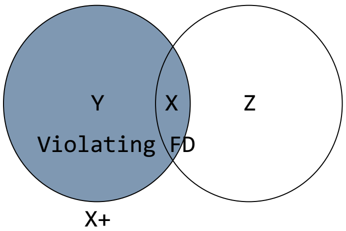


11

SIT Internal

BCNF Decomposition Algorithm

- **Cannot** just **arbitrarily break** a relation schema.
  - **Start** with the **FD** that **violates** BCNF, or the FD that is not a super-key.
  - Attributes are broken into two **overlapping** relation schema.
  - One is all the attributes in the violating FD.
- Normalize(R):
  - Given all FDs X -> A.
  - Find X -> A, s.t.: A not in X && X+ ≠ [all attri.]
  - If not found, then "R is in BCNF". Return.
  - Let Y = X+ - X.
  - let Z = [all attributes] - X+.
  - decompose R into R1(X, Y) and R2(X, Z).
  - Normalize(R1) and Normalize(R2).



R(A1, …, An, B1, …, Bm, C1, …, Cp)

R1(A1, …, An, B1, …, Bm)

R2(A1, …, An, C1, …, Cp)

12

SIT Internal

BCNF Decomposition – Example

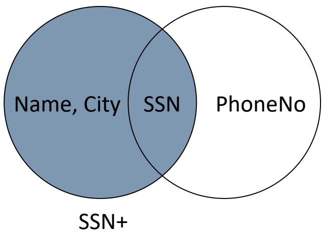
- Given only 1 FD: SSN -> Name, City.
- Violating FD (SSN+ = SSN, Name, City).
- Use (SSN -> Name, City) to split.
- After decomposition.
  - SSN+ = {SSN, Name, City} for R1.

Name	SSN	PhoneNo	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Westfield

SSN	PhoneNo
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Anomalies
- Redundancy?
- Update?
- Delete?



13

SIT Internal

More about Decomposition and Summary

- Decomposition: **not perfect**; sometimes create more problems.
  - **Lossless-join** property (BCNF ✓): recover instance of decomposed relation from instances of smaller relations -> no turning back?
  - **Dependency-preservation** property (BCNF ✗): enforce constraint on original relation by enforcing constraints on smaller relations.
  - **Performance** consideration: **join**.
- Storyline:
  - Imperfect design -> **redundancy**.
  - Redundancy -> **decomposition**.
  - Decomposition depends on **normal form**.
  - BCNF based on **FD**.
  - Given some FDs -> closure.
  - **Closure** -> decomposition -> better design.

S1 • Requirement Analysis
S2 • Conceptual Database Design
S3 • Logical Database Design
S4 • **Refine the Schemas**
S5 • Physical Database Design
S6 • Application & Security Design

14