Database Management Systems

Module 21: Relational Database Design/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Week Recap

- Discussed relational algebra with examples
- Introduced tuple relational and domain relational calculus
- Illustrated equivalence of algebra and calculus
- Introduced the Design Process for Database Systems
- Elucidated the E-R Model for real world representation with entities, entity sets, attributes, and relationships
- Illustrated ER Diagram notation for ER Models
- Discussed translation of ER Models to Relational Schema and extended features of ER Model
- Deliberated on various design issues

Module Objectives

- To identify the features of good relational design
- To familiarize with the First Normal Form

Module Outline

- Features of Good Relational Design
- Atomic Domains and First Normal Form

Features of Good Relational Design

Good Relational Design

- Reflects *real-world structure* of the problem
- Can represent *all expected data* over time
- Avoids *redundant storage* of data items
- Provides *efficient access* to data
- Supports the *maintenance of data integrity* over time
- *Clean, consistent, and easy* to understand
- Note: *These objectives are sometimes contradictory!*

What is a Good Schema?

instructor_with_department

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

- *ID*: Key
- *building*, *budget*: Redundant Information
- *name*, *salary*, *dept_name*: No Redundant Information

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

department

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

What is a Good Schema? (2)

- Consider combining relations
  - *sec_class*(*sec_id*, *building*, *room_number*) and
  - *section*(*course_id*, *sec_id*, *semester*, *year*)

  into one relation
  - *section*(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*)
- No repetition in this case

Redundancy and Anomaly

- **Redundancy**: having multiple copies of same data in the database.
  - This problem arises when a database is not normalized
  - It leads to anomalies
- **Anomaly**: inconsistencies that can arise due to data changes in a database with insertion, deletion, and update
  - These problems occur in poorly planned, un-normalised databases where all the data is stored in one table (a flat-file database)

  There can be three kinds of anomalies
  - *Insertions Anomaly*
  - *Deletion Anomaly*
  - *Update Anomaly*

Redundancy and Anomaly (2)

- **Insertions Anomaly**
  - When the insertion of a data record is not possible without adding some additional unrelated data to the record
  - We cannot add an Instructor in *instructor_with_department* if the *department* does not have a *building* or *budget*
- **Deletion Anomaly**
  - When deletion of a data record results in losing some unrelated information that was stored as part of the record that was deleted from a table
  - We delete the last Instructor of a Department from *instructor_with_department*, we lose *building* and *budget* information
- **Update Anomaly**
  - When a data is changed, which could involve many records having to be changed, leading to the possibility of some changes being made incorrectly
  - When the *budget* changes for a Department having large number of Instructors in *instructor_with_department* application may miss some of them

Redundancy and Anomaly (3)

- We have observed the following:
  - **Redundancy** $\Rightarrow$ **Anomaly**
  - Relations *instructor* and *department* is better than *instructor_with_department*
- What causes redundancy?
  - **Dependency** $\Rightarrow$ **Redundancy**
  - *dept_name* uniquely decides *building* and *budget*. A department cannot have two different budget or building. So *building* and *budget* **depends on** *dept_name*
- How to remove, or at least minimize, redundancy?
  - Decompose (partition) the relation into smaller relations
  - *instructor_with_department* can be decomposed into *instructor* and *department*
  - **Good Decomposition** $\Rightarrow$ **Minimization of Dependency**
- Is every decomposition good?
  - No. It needs to preserve information, honour the dependencies, be efficient etc.
  - Various schemes of normalization ensure good decomposition
  - **Normalization** $\Rightarrow$ **Good Decomposition**

Decomposition

- Suppose we had started with *inst_dept*. How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule "if there were a schema *(dept_name, building, budget)*, then *dept_name* would be a candidate key"
- Denote as a **functional dependency**: $dept\_name \rightarrow building, budget$
- In *inst_dept*, because *dept_name* is not a candidate key, the *building* and *budget* of a *department* may have to be repeated.
  - This indicates the need to decompose *inst_dept*

Decomposition (2)

- Not all decompositions are good
- Suppose we decompose
  employee(ID, name, street, city, salary) into
  - *employee1 (<u>ID</u>, name)*
  - *employee2 (name, street, city, salary)*
- Note that if *name* can be duplicate, then *employee*2 is a weak entity set and cannot exist without an identifying relationship
- Consequently, this decomposition cannot preserve the information
- The next slide shows how we lose information – we cannot reconstruct the *original employee* relation – and so, this is a **lossy decomposition**.

Decomposition (3): Lossy Decomposition

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

Duplicate names cause the problem here.

ID	*name*
⋮	
57766	Kim
98776	Kim
⋮	

name	*street*	*city*	*salary*
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	*name*	*street*	*city*	*salary*
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Decomposition (4): Lossless-Join Decomposition

- **Lossless Join Decomposition**
- Decomposition of $R = (A, B, C)$
  $R_1 = (A, B),\ R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\prod_{A,B}(r)$

B	C
1	A
2	B

$\prod_{B,C}(r)$

$\prod_{A,B}(r) \bowtie \prod_{B,C}(r)$

A	B	C
α	1	A
β	2	B

Decomposition (5): Lossless-Join Decomposition

- **Lossless Join Decomposition** is a decomposition of a relation R into relations R_1, R_2 such that if we perform natural join of two smaller relations it will return the original relation

$$R_1 \cup R_2 = R, R_1 \cap R_2 \neq \phi$$

$$\forall r \in R, r_1 = \sqcap_{R_1}(r), r_2 = \sqcap_{R_2}(r)$$

$$r_1 \bowtie r_2 = r$$

- This is effective in removing redundancy from databases while preserving the original data
- In other words by lossless decomposition it becomes feasible to reconstruct the relation R from decomposed tables R_1 and R_2 by using Joins

Atomic Domains and First Normal Form

First Normal Form (1NF)

- A domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **First Normal Form (INF)** if
  - the domains of all attributes of R are *atomic*
  - the value of each attribute contains only a *single value* from that domain
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - *We assume all relations are in first normal form*

First Normal Form (2)

- *Atomicity* is actually a property of how the elements of the domain are used
  - Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - **If the first two characters are extracted to find the department, the domain of roll numbers is not atomic**
  - *Doing so is a bad idea*
    - Leads to encoding of information in application program rather than in the database

First Normal Form (3)

- The following is not in 1NF

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

  - A telephone number is composite
  - Telephone number is multi-valued

First Normal Form (4)

- Consider:

Customer

Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

  - is in 1NF if telephone number is not considered composite
  - However, conceptually, we have two attributes for the same concept
    - Arbitrary and meaningless ordering of attributes
    - How to search telephone numbers
    - Why only two numbers?

First Normal Form (5)

- Is the following in 1NF?

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

  - Duplicated information
  - ID is no more the key. Key is (ID, Telephone Number)

First Normal Form (6)

- Better to have 2 relations:

Customer Name

Customer ID	First Name	Surname
123	Pooja	Singh
456	San	Zhang
789	John	Doe

Customer Telephone Number

Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
456	(555) 403-1659 Ext. 53
456	182-929-2929
789	555-808-9633

  - One-to-Many relationship between parent and child relations
  - Incidentally, satisfies 2NF and 3NF
- Decomposition helps to attain 1NF for the embedded one-to-many relationship

Module Summary

- Identified the features of good relational design
- Familiarized with the First Normal Form

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 22: Relational Database Design/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Identified the features of good relational design
- Familiarized with the First Normal Form

Module Objectives

- To Introduce Functional Dependencies

Module Outline

- Functional Dependencies

Functional Dependencies

Goal: Devise a Theory for Good Relations

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- The theory is based on:
  - Functional dependencies
  - Multivalued dependencies
  - Other dependencies

Functional Dependencies

- Constraints on the set of legal relations
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a *key*

Functional Dependencies (2)

- Let R be a relation schema

  $\alpha \subseteq R$ and $\beta \subseteq R$
- The functional dependency or FD

  $\alpha \rightarrow \beta$

  holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A, B)$ with the following instance of r.

A	B
1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold. So we cannot have tuples like (2, 4), or (3, 5), or (4, 7) added to the current instance.

Functional Dependencies (3)

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$ and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  $inst\_dept(\underline{ID}, name, salary, \underline{dept\_name}, building, budget)$
- We expect these functional dependencies to hold:

  $dept\_name \rightarrow building$

  $dept\_name \rightarrow budget$

  $ID \rightarrow budget$

  but would not expect the following to hold:

  $dept\_name \rightarrow salary$

Functional Dependencies (4)

- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation r is legal under a set F of functional dependencies, we say that r **satisfies** F
  - specify constraints on the set of legal relations
    - We say that F **holds on** R if all legal relations on R satisfy the set of functional dependencies F
- **Note**: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances
  - For example, a specific instance of instructor may, by chance, satisfy

    $$name \rightarrow ID$$
  - In such cases we do not say that F holds on R

Functional Dependencies (5)

- A functional dependency is trivial if it is satisfied by all instances of a relation
  - Example:
    - $ID, name \rightarrow ID$
    - $name \rightarrow name$
- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.

Functional Dependencies (6)

- Functional dependencies are:

StudentID	Semester	Lecture	TA
1234	6	Numerical Methods	John
1221	4	Numerical Methods	Smith
1234	6	Visual Computing	Bob
1201	2	Numerical Methods	Peter
1201	2	Physics II	Simon

  - $StudentID \rightarrow Semester$
    $StudentID, Lecture \rightarrow TA$
    $\{StudentID, Lecture\} \rightarrow \{TA, Semester\}$

Functional Dependencies (7)

- Functional dependencies are:

Employee ID	Employee Name	Department ID	Department Name
0001	John Doe	1	Human Resources
0002	Jane Doe	2	Marketing
0003	John Smith	1	Human Resources
0004	Jane Goodall	3	Sales

  - $EmployeeID \rightarrow EmployeeName$
    $EmployeeID \rightarrow DepartmentID$
    $DepartmentID \rightarrow DepartmentName$

Functional Dependencies (8): Armstrong's Axioms

- Given a set of Functional Dependencies F, we can infer new dependencies by the **Armstrong's Axioms**:
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These axioms can be repeatedly applied to generate new FDs and added to F
- A new FD obtained by applying the axioms is said to the **logically implied** by F
- The process of generations of FDs terminate after finite number of steps and we call it the **Closure Set** F^+ for FDs F. This is the set of **all** FDs logically implied by F
- Clearly, $F \subseteq F^+$
- These axioms are
  - **Sound** (generate only functional dependencies that actually hold), and
  - **Complete** (eventually generate all functional dependencies that hold)
- Prove the axioms from definitions of FDs
- Prove the soundness and completeness of the axioms

Functional Dependencies (9): Closure of a Set of FDs

- $F = \{A \rightarrow B, B \rightarrow C\}$
- $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Module Summary

- Introduced the notion of Functional Dependencies

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with “PPD”.

Database Management Systems

Module 23: Relational Database Design/3

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced the notion of Functional Dependencies

Module Objectives

- To develop the theory of functional dependencies
- To understand how a schema can be decomposed for a 'good' design using functional dependencies

Module Outline

- Functional Dependency Theory
- Decomposition Using Functional Dependencies

Functional Dependency Theory

Functional Dependencies: Armstrong's Axioms

- Given a set of Functional Dependencies F, we can infer new dependencies by the **Armstrong's Axioms**:
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- These axioms can be repeatedly applied to generate new FDs and added to F
- A new FD obtained by applying the axioms is said to the **logically implied** by F
- The process of generations of FDs terminate after finite number of steps and we call it the **Closure Set** F^+ for FDs F. This is the set of **all** FDs logically implied by F
- Clearly, $F \subseteq F^+$
- These axioms are
  - **Sound** (generate only functional dependencies that actually hold), and
  - **Complete** (eventually generate all functional dependencies that hold)
- Prove the axioms from definitions of FDs
- Prove the soundness and completeness of the axioms

Functional Dependencies (2): Closure of a Set FDs

- $F = \{A \rightarrow B, B \rightarrow C\}$
- $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Functional Dependencies (3): Closure of a Set FDs

- $R = (A, B, C, G, H, I)$

$$F = \{A \rightarrow B \\ A \rightarrow C \\ CG \rightarrow H \\ CG \rightarrow I \\ B \rightarrow H\}$$

- Some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ and $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ with CG to infer $CG \rightarrow CGI$, and augmenting $CG \rightarrow H$ with I to infer $CGI \rightarrow HI$, and then transitivity

Functional Dependencies (4): Closure of a Set FDs: Computing F^+

- To compute the closure of a set of functional dependencies F:

  $F^+ \leftarrow F$
  repeat
      for each functional dependency f in F^+
          apply reflexivity and augmentation rules on f
          add the resulting functional dependencies to F^+
      for each pair of functional dependencies f_1 and f_2 in F^+
          if f_1 and f_2 can be combined using transitivity
              then add the resulting functional dependency to F^+
  until F^+ does not change any further

- **Note**: We shall see an alternative procedure for this task later

Functional Dependencies (5): Armstrong's Axioms: Derived Rules

- Additional Derived Rules:
  - **Union**: if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
  - **Decomposition**: if $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds
  - **Pseudotransitivity**: if $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds
- The above rules can be inferred from basic Armstrong's axioms (and hence are not included in the basic set). They can be proven independently too
  - **Reflexivity**: if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
  - **Augmentation**: if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
  - **Transitivity**: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- Prove the Rules from:
  - Basic Axioms
  - The definitions of FDs

Functional Dependencies (6): Closure of Attribute Sets

- Given a set of attributes α, define the **closure** of α **under** F (denoted by α^+) as the set of attributes that are functionally determined by α under F
- Algorithm to compute α^+, the closure of α under F

  result $\leftarrow \alpha$
  while (changes to *result*) **do**
      for each $\beta \rightarrow \gamma$ **in** F **do**
          begin
              if $\beta \subseteq$ result **then** *result* $\leftarrow$ *result* $\cup\ \gamma$
          end

Functional Dependencies (7): Closure of Attribute Sets: Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$
- $(AG)^+$
  a) result = AG
  b) result = ABCG ($A \to C$ and $A \to B$)
  c) result = ABCGH ($CG \to H$ and $CG \subseteq AGBC$)
  d) result = ABCGHI ($CG \to I$ and $CG \subseteq AGBCH$)
- Is AG a candidate key?
  a) Is AG a super key?
     i) Does $AG \to R$? == Is $(AG)^+ \supseteq R$
  b) Is any subset of AG a superkey?
     i) Does $A \to R$? == Is $(A)^+ \supseteq R$
     ii) Does $G \to R$? == Is $(G)^+ \supseteq R$

Functional Dependencies (7): Closure of Attribute Sets: Use

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Decomposition using Functional Dependencies

BCNF: Boyce-Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of FDs if for all FDs in F^+ of the form

  $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R
- Example schema *not in* BCNF:

  instr_dept (*ID*, *name*, *salary*, *dept_name*, *building*, *budget*)
- because the non-trivial dependency *dept_name* $\rightarrow$ *building*, *budget* holds on *instr_dept*, but *dept_name* is not a superkey

BCNF (2): Decomposition

- If in schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF, we decompose R into:
  - $\alpha \cup \beta$
  - $(R - (\beta - \alpha))$
- In our example,
  - $\alpha = dept\_name$
  - $\beta = building, budget$
  - $dept\_name \rightarrow building,\ budget$

  inst_dept is replaced by
  - $(\alpha \cup \beta) = (dept\_name,\ building,\ budget)$
    - $dept\_name \rightarrow building,\ budget$
  - $(R - (\beta - \alpha)) =(ID,\ name,\ salary,\ dept\_name)$
    - $ID \rightarrow name,\ salary,\ dept\_name$

BCNF (3): Lossless Join

- If we decompose a relation R into relations R_1 and R_2:
  - Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
  - Decomposition is lossless if $R_1 \bowtie R_2 = R$
- To check for lossless join decomposition using FD set, following must hold:
  - Union of Attributes of R_1 and R_2 must be equal to attribute of R

$$R_1 \cup R_2 = R$$

  - Intersection of Attributes of R_1 and R_2 must not be NULL

$$R_1 \cap R_2 \neq \Phi$$

  - Common attribute must be a key for at least one relation (R_1 or R_2)

$$R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2$$

- Prove that BCNF ensures Lossless Join

BCNF (4): Dependency Preservation

- Constraints, including FDs, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- It is not always possible to achieve both BCNF and dependency preservation. Consider:
  - $R = CSZ$, $F = \{CS \rightarrow Z, Z \rightarrow C\}$
  - Key $= CS$
  - $CS \rightarrow Z$ satisfies BCNF, but $Z \rightarrow C$ violates
  - Decompose as: $R_1 = ZC, R_2 = CSZ - (C - Z) = SZ$
  - $R_1 \cup R_2 = CSZ = R$, $R_1 \cap R_2 = Z \neq \Phi$, and $R_1 \cap R_2 = Z \rightarrow ZC = R_1$. So it has *lossless join*
  - However, we cannot check $CS \rightarrow Z$ without doing a join. Hence it is not *dependency preserving*
- We consider a weaker normal form, known as **Third Normal Form (3NF)**

3NF: Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:

  $$\alpha \rightarrow \beta \in F^+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (that is, $\beta \subseteq \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R
    (Nore: Each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in "good" form
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving

Problems with Decomposition

There are three potential problems to consider:

- May be impossible to reconstruct the original relation! (Lossiness)
- Dependency checking may require joins
- Some queries become more expensive
  - What is the building for an instructor?

Tradeoff: Must consider these issues vs. redundancy

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation

  inst_info (ID, child_name, phone)

  - where an instructor may have more than one phone and can have multiple children

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	Willian	512-555-4321

inst_info

How good is BCNF? (2)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – that is, if we add a phone 981-992-3443 to 99999, we need to add two tuples

(99999, David, 981-992-3443)
(99999, William, 981-992-3443)

How good is BCNF? (3)

- Therefore, it is better to decompose *inst_info* into:

inst_child

ID	*child_name*
99999	David
99999	William

inst_phone

ID	*phone*
99999	512-555-1234
99999	512-555-4321

- This suggests the need for higher normal forms, such as the Fourth Normal Form (4NF)

Module Summary

- Introduced the theory of functional dependencies
- Discussed issues in "good" design in the context of functional dependencies

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".

Database Management Systems

Module 24: Relational Database Design/4

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Introduced the theory of functional dependencies
- Discussed issues in "good" design in the context of functional dependencies

Module Objectives

- To Learn Algorithms for Properties of Functional Dependencies

Module Outline

- Algorithms for Functional Dependencies

Algorithms for Functional Dependencies

Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B,\ A \rightarrow C,\ CG \rightarrow H,\ CG \rightarrow I,\ B \rightarrow H\}$
- $(AG)^+$
  a) $result = AG$
  b) $result = ABCG$ $\quad (A \rightarrow C$ and $A \rightarrow B)$
  c) $result = ABCGH$ $\quad (CG \rightarrow H$ and $CG \subseteq AGBC)$
  d) $result = ABCGHI$ $\quad (CG \rightarrow I$ and $CG \subseteq AGBCH)$
- Is AG a candidate key?
  a) Is AG a super key?
     i) Does $AG \rightarrow R? ==$ Is $(AG)^+ \supseteq R$
  b) Is any subset of AG a superkey?
     i) Does $A \rightarrow R? ==$ Is $(A)^+ \supseteq R$
     ii) Does $G \rightarrow R? ==$ Is $(G)^+ \supseteq R$

Attribute Set Closure: Uses

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
  - That is, we compute α^+ by using attribute closure, and then check if it contains β.
  - Is a simple and cheap test, and very useful
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Extraneous Attributes

- Consider a set F of FDs and the FD $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β if $A \in \beta$ and the set of FDs $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (that is, the result of dropping B from $AB \rightarrow C$).
  - $A^+ = AC$ in $\{A \rightarrow C, AB \rightarrow C\}$
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C
  - $AB^+ = ABCD$ in $\{A \rightarrow C, AB \rightarrow D\}$

Extraneous Attributes (2): Tests

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
- To test if attribute $A \in \alpha$ is extraneous in α
  a) Compute $(\{\alpha\} - A)^+$ using the dependencies in F
  b) Check that $(\{\alpha\} - A)^+$ contains β; if it does, A is extraneous in α
- To test if attribute $A \in \beta$ is extraneous in β
  a) Compute α^+ using only the dependencies in
  $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  b) Check that α^+ contains A; if it does, A is extraneous in β

Equivalence of Sets of Functional Dependencies

- Let F & G are two functional dependency sets.
  - These two sets F & G are equivalent if $F^+ = G^+$. That is: $(F^+ = G^+) \Leftrightarrow (F^+ \Rightarrow G \text{ and } G^+ \Rightarrow F)$
  - Equivalence means that every functional dependency in F can be inferred from G, and every functional dependency in G an be inferred from F
- F and G are equal only if
  - F covers G: Means that all functional dependency of G are logically numbers of functional dependency set $F \Rightarrow F^+ \supseteq G$.
  - G covers F: Means that all functional dependency of F are logically members of functional dependency set $G \Rightarrow G^+ \supseteq F$.

Condition	CASES			
F Covers G	True	True	False	False
G Covers F	True	False	True	False
Result	F=G	F⊃G	G⊃F	No Comparison

Canonical Cover

- Sets of FDs may have redundant dependencies that can be inferred from the others
- Can we have some kind of "optimal" or "minimal" set of FDs wto work with?
- A **Canonical Cover** for F is a set of dependencies F_c such that ALL the following properties are satisfied:
  - $F^+ = F_c^+$. Or,
    - F logically implies all dependencies in F_c
    - F_c logically implies all dependencies in F
  - No functional dependency in F_c contains an extraneous attribute
  - Each left side of functional dependency in F_c is unique. That is, there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in such that $\alpha_1 \rightarrow \alpha_2$
- Intuitively, a **Canonical cover** of F is a **minimal** set of FDs
  - Equivalent to F
  - Having no redundant FDs
  - No redundant parts of FDs
- **Minimal / Irreducible Set of Functional Dependencies**

Canonical Cover (2): Example

- For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
- Parts of a functional dependency may be redundant
  - For example: on RHS: $\boldsymbol{\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}}$ can be simplified to $\boldsymbol{\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}}$
  - In the forward: (1) $A \rightarrow CD \Rightarrow A \rightarrow C$ and $A \rightarrow D$
    (2) $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$
  - In the reverse: (1) $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$
    (2) $A \rightarrow C, A \rightarrow D \Rightarrow A \rightarrow CD$
  - For example: on LHS: $\boldsymbol{\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}}$ can be simplified to $\boldsymbol{\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}}$
  - In the forward: (1) $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C \Rightarrow A \rightarrow AC$
    (2) $A \rightarrow AC, AC \rightarrow D \Rightarrow A \rightarrow D$
  - In the reverse: $A \rightarrow D \Rightarrow AC \rightarrow D$

Canonical Cover (3): RHS

- $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\} \Rightarrow \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
  - (1) $A \rightarrow CD \Rightarrow A \rightarrow C$ and $A \rightarrow D$
    (2) $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$
  - $A^+ = ABCD$
- $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\} \Rightarrow \{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$
  - $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$
  - $A \rightarrow C, A \rightarrow D \Rightarrow A \rightarrow CD$
  - $A^+ = ABCD$

Canonical Cover (4): LHS

- $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\} \Rightarrow \{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
  - $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C \Rightarrow A \rightarrow AC$
  - $A \rightarrow AC, AC \rightarrow D \Rightarrow A \rightarrow D$
  - $A^+ = ABCD$
- $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\} \Rightarrow \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$
  - $A \rightarrow D \Rightarrow AC \rightarrow D$
  - $AC^+ = ABCD$

Canonical Cover (5)

- To compute a canonical cover for F:
  repeat
  Use the union rule to replace any dependencies in F
  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$
  Find a functional dependency $\alpha \rightarrow \beta$ with an
  extraneous attribute either in α or in β
  /* Note: test for extraneous attributes done using F_c, not F */
  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until F does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Canonical Cover (6): Example

- $R = (A, B, C)$
  $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - ▷ Yes: in fact, $B \rightarrow C$ is already present!
  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - ▷ Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$.
      - Can use attribute closure of A in more complex cases
- The canonical cover is: $A \rightarrow B$, $B \rightarrow C$

Practice Problems on Functional Dependencies

- **Find if a given functional dependency is implied from a set of Functional Dependencies:**
  a) For: $A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH, ABH \rightarrow BD, DH \rightarrow BC$
     i) Check: $BCD \rightarrow H$
     ii) Check: $AED \rightarrow C$
  b) For: $AB \rightarrow CD, AF \rightarrow D, DE \rightarrow F, C \rightarrow G, F \rightarrow E, G \rightarrow A$
     i) Check: $CF \rightarrow DF$
     ii) Check: $BG \rightarrow E$
     iii) Check: $AF \rightarrow G$
     iv) Check: $AB \rightarrow EF$
  c) For: $A \rightarrow BC, B \rightarrow E, CD \rightarrow EF$
     i) Check: $AD \rightarrow F$

Practice Problems on Functional Dependencies (2)

- **Find Super Key using Functional Dependencies:**
  a) Relational Schema $R(ABCDE)$. Functional dependencies: $AB \rightarrow C, DE \rightarrow B, CD \rightarrow E$
  b) Relational Schema $R(ABCDE)$. Functional dependencies: $AB \rightarrow C, C \rightarrow D, B \rightarrow EA$

Practice Problems on Functional Dependencies (3)

- **Find Candidate Key using Functional Dependencies:**
  a) Relational Schema $R(ABCDE)$. Functional dependencies: $AB \rightarrow C, DE \rightarrow B, CD \rightarrow E$
  b) Relational Schema $R(ABCDE)$. Functional dependencies: $AB \rightarrow C, C \rightarrow D, B \rightarrow EA$

Practice Problems on Functional Dependencies (4)

- **Find Prime and Non Prime Attributes using Functional Dependencies:**
  a) $R(ABCDEF)$ having FDs $\{AB \rightarrow C, C \rightarrow D, D \rightarrow E, F \rightarrow B, E \rightarrow F\}$
  b) $R(ABCDEF)$ having FDs $\{AB \rightarrow C, C \rightarrow DE, E \rightarrow F, C \rightarrow B\}$
  c) $R(ABCDEFGHIJ)$ having FDs $\{AB \rightarrow C, A \rightarrow DE, B \rightarrow F, F \rightarrow GH, D \rightarrow IJ\}$
  d) $R(ABDLPT)$ having FDs $\{B \rightarrow PT, A \rightarrow D, T \rightarrow L\}$
  e) $R(ABCDEFGH)$ having FDs $\{E \rightarrow G, AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A\}$
  f) $R(ABCDE)$ having FDs $\{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$
  g) $R(ABCDEH)$ having FDs $\{A \rightarrow B, BC \rightarrow D, E \rightarrow C, D \rightarrow A\}$

- **Prime Attributes**: Attribute set that belongs to any candidate key are called Prime Attributes
  - It is union of all the candidate key attribute: $\{CK_1 \cup CK_2 \cup CK_3 \cup \cdots\}$
  - If Prime attribute determined by other attribute set, then more than one candidate key is possible.
  - For example, If A is Candidate Key, and $X \rightarrow A$, then, X is also Candidate Key.
- **Non Prime Attribute**: Attribute set does not belong to any candidate key are called Non Prime Attributes

Practice Problems on Functional Dependencies (5)

- **Check the Equivalence of a Pair of Sets of Functional Dependencies:**
  a) Consider the two sets F and G with their FDs as below :
     i) $F : A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H$
     ii) $G : A \rightarrow CD, E \rightarrow AH$
  b) Consider the two sets P and Q with their FDs as below :
     i) $P : A \rightarrow B, AB \rightarrow C, D \rightarrow ACE$
     ii) $Q : A \rightarrow BC, D \rightarrow AE$

Practice Problems on Functional Dependencies (6)

- **Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:**
  a) $AB \rightarrow CD, BC \rightarrow D$
  b) $ABCD \rightarrow E, E \rightarrow D, AC \rightarrow D, A \rightarrow B$

Module Summary

- Studied Algorithms for Properties of Functional Dependencies

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with “PPD”.

Database Management Systems

Module 25: Relational Database Design/5

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

Module Recap

- Studied Algorithms for Properties of Functional Dependencies

Module Objectives

- To Understand the Characterizations for Lossless Join Decomposition
- To Understand the Characterizations for Dependency Preservation

Module Outline

- Lossless Join Decomposition
- Dependency Preservation

Lossless Join Decomposition

Lossless Join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

To Identify whether a decomposition is lossy or lossless, it must satisfy the following conditions:

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \phi$ and
- $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$

Lossless Join Decomposition (2): Example

- Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**
- Having dependencies: **S#** → **Sname, S#** → **City, (S#, P#)** → **Qty**
- Decompose as: **Supplier(S#, Sname, City, Qty): Parts(P#, Qty)**
- Take Natural Join to reconstruct: **Supplier** $\bowtie$ **Parts**

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

P#	Qty
301	20
500	50
20	10
400	40
301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
5	**Nick**	**NY**	**20**	**10**
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10
2	**Steve**	**Boston**	**301**	**10**

- We get extra tuples! **Join is Lossy!**
- Common attribute **Qty** is not a superkey in **Supplier** or in **Parts**
- Does not preserve **(S#, P#)** → **Qty**

Lossless Join Decomposition (3): Example

- Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**
- Having dependencies: **S#** $\rightarrow$ **Sname, S#** $\rightarrow$ **City, (S#, P#)** $\rightarrow$ **Qty**
- Decompose as: **Supplier(S#, Sname, City): Parts(S#, P#, Qty)**
- Take Natural Join to reconstruct: **Supplier** $\bowtie$ **Parts**

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City
3	Smith	London
5	Nick	NY
2	Steve	Boston
5	Nick	NY
5	Nick	NY

S#	P#	Qty
3	301	20
5	500	50
2	20	10
5	400	40
5	301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

- We get back the original relation. **Join is Lossless.**
- Common attribute **S#** is a superkey in **Supplier**
- Preserves all dependencies

Lossless Join Decomposition (4): Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in two different ways
- $R_1 = (A, B), R_2 = (B, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B), R_2 = (A, C)$
  - Lossless-join decomposition:
    $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Practice Problems on Lossless Join

- **Check if the decomposition of R into D is lossless:**
  a) $R(ABC) : F = \{A \rightarrow B, A \rightarrow C\}$. $D = R_1(AB), R_2(BC)$
  b) $R(ABCDEF) : F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, E \rightarrow F\}$.
     $D = R_1(AB), R_2(BCD), R_3(DEF)$
  c) $R(ABCDEF) : F = \{A \rightarrow B, C \rightarrow DE, AC \rightarrow F\}$. $D = R_1(BE), R_2(ACDEF)$
  d) $R(ABCDEG) : F = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$
     i) $D1 = R_1(AB), R_2(BC), R_3(ABDE), R_4(EG)$
     ii) $D2 = R_1(ABC), R_2(ACDE), R_3(ADG)$
  e) $R(ABCDEFGHIJ) : F = \{AB \rightarrow C, B \rightarrow F, D \rightarrow IJ, A \rightarrow DE, F \rightarrow GH\}$
     i) $D1 = R_1(ABC), R_2(ADE), R_3(BF), R_4(FGH), R_5(DIJ)$
     ii) $D2 = R_1(ABCDE), R_2(BFGH), R_3(DIJ)$
     iii) $D3 = R_1(ABCD), R_2(DE), R_3(BF), R_4(FGH), R_5(DIJ)$

Dependency Preservation

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i
  - A decomposition is **dependency preserving**, if

$$(F_1 \cup F_2 \cup \cdots \cup F_n)^+ = F^+$$

  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Let R be the original relational schema having FD set F. Let R_1 and R_2 having FD set F_1 and F_2 respectively, are the decomposed sub-relations of R. The decomposition of R is said to be preserving if

- $F_1 \cup F_2 \equiv F$ {Decomposition Preserving Dependency}
- If $F_1 \cup F_2 \subset F$ {Decomposition NOT Preserving Dependency} and
- $F_1 \cup F_2 \supset F$ {this is not possible}

Dependency Preservation (2): Testing

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $D = \{R_1, R_2, \ldots, R_n\}$ we apply the following test (with attribute closure done with respect to F)
- The **restriction** of F^+ to R_i is the set of all functional dependencies in F^+ that include only attributes of R_i.
  - compute F^+;
    for each schema R_i in D **do**
        begin
            F_i = the restriction of F^+ to R_i;
        end
    $F' = \phi$
    for each restriction F_i **do**
        begin
            $F' = F' \cup F_i$
        end
    compute F'^+;
    if $(F'^+ = F^+)$ **then** return (true)
                    else return (false);
- The procedure for checking dependency preservation takes exponential time to compute F^+ and $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

Dependency Preservation (3): Example

- **R** (A, B, C, D, E, F)
  F $= \{A \rightarrow BCD, A \rightarrow EF, BC \rightarrow AD, BC \rightarrow E, BC \rightarrow F, B \rightarrow F, D \rightarrow E\}$
- Decomposition: **R1**(A, B, C, D) **R2**(B, F) **R3**(D, E)
  - $A \rightarrow BCD$, $BC \rightarrow AD$ are preserved on table R1
  - $B \rightarrow F$ is preserved on table R2
  - $D \rightarrow E$ is preserved on table R3
  - We have to check whether the remaining FDs: **A→E**, **A→ F**, **BC→E**, **BC→ F** are preserved or not.

R1	R2	R3
F_1={**A** $\rightarrow ABCD$, **B** $\rightarrow B$, **C** $\rightarrow C$, **D** $\rightarrow D$, **AB** $\rightarrow ABCD$, **BC** $\rightarrow ABCD$, **CD** $\rightarrow CD$, **AD** $\rightarrow ABCD$ **ABC** $\rightarrow ABCD$, **ABD** $\rightarrow ABCD$, **ACD** $\rightarrow ABCD$ **BCD** $\rightarrow ABCD$}	F_2={**B** $\rightarrow BF$, **F** $\rightarrow F$}	F_3={**D** $\rightarrow DE$, **E** $\rightarrow E$}

- $F' = F_1 \cup F_2 \cup F_3$.
- Checking for: **A** $\rightarrow E$, **A** $\rightarrow F$ in F'^{+}
  - ▷ $A \rightarrow D$ (from R1), $D \rightarrow E$ (from R3) : **A→ E** (By Transitivity)
  - ▷ $A \rightarrow B$ (from R1), $B \rightarrow F$ (from R2) : **A→ F** (By Transitivity)
- Checking for: **BC** $\rightarrow E$, **BC** $\rightarrow F$ in F'^{+}
  - ▷ $BC \rightarrow D$ (from R1), $D \rightarrow E$ (from R3) : **BC→ E** (By Transitivity)
  - ▷ $B \rightarrow F$ (from R2) : **BC→ F** (By Augmentation)

Hence all dependencies are preserved.

Dependency Preservation (4): Example

- **R** (A, B, C, D)
  F $= \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$
- Decomposition: **R1**(A, B) **R2**(B, C) **R3**(C, D)
  - $A \rightarrow B$ is preserved on table R1
  - $B \rightarrow C$ is preserved on table R2
  - $C \rightarrow D$ is preserved on table R3
  - We have to check whether the one remaining FD: **D→A** is preserved or not.

R1	R2	R3
$F_1 = \{\mathbf{A} \rightarrow AB, \mathbf{B} \rightarrow BA\}$	$F_2 = \{\mathbf{B} \rightarrow BC, \mathbf{C} \rightarrow CB\}$	$F_3 = \{\mathbf{C} \rightarrow CD, \mathbf{D} \rightarrow DC\}$

- $F' = F_1 \cup F_2 \cup F_3$.
- Checking for: $\mathbf{D} \rightarrow A$ in F'^+
  - $D \rightarrow C$ (from R3), $C \rightarrow B$ (from R2), $B \rightarrow A$ (from R1) : **D→ A** (By Transitivity)
    Hence all dependencies are preserved.

Dependency Preservation (5): Testing

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \cdots, R_n$ we apply the following test (with attribute closure done with respect to F)
  - $result = \alpha$
    while (changes to result) do
        for each R_i in the decomposition
            $t = (result \cap R_i)^+ \cap Ri$
            $result = result \cup t$
  - If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.
- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \cdots \cup F_n)^+$

Dependency Preservation (6): Example

- $R(ABCDEF) :.\ F = \{A \rightarrow BCD, A \rightarrow EF, BC \rightarrow AD, BC \rightarrow E, BC \rightarrow F, B \rightarrow F, D \rightarrow E\}$
- $Decomp = \{ABCD, BF, DE\}$
- On projections:

ABCD (R1)	BF (R2)	DE (R3)
A → BCD BC → AD	B → F	D → E

- Need to check for: $\cancel{A \rightarrow BCD}, \boldsymbol{A \rightarrow EF}, \cancel{BC \rightarrow AD}, \boldsymbol{BC \rightarrow E}, \boldsymbol{BC \rightarrow F}, \cancel{B \rightarrow F}, \cancel{D \rightarrow E}$
- $(BC)+/F1 = ABCD$. $(ABCD)+/F2 = ABCDF$. $(\boldsymbol{ABCDF})+/F3 = \boldsymbol{ABCDEF}$. Preserves $\boldsymbol{BC \rightarrow E, BC \rightarrow F}$
  $BC \rightarrow AD$ $(R1)$, $AD \rightarrow E$ $(R3)$ implies $BC \rightarrow E$
  $B \rightarrow F$ $(R2)$ implies $BC \rightarrow F$
- $(A)+/F1 = ABCD$. $(ABCD)+/F2 = ABCDF$. $(\boldsymbol{ABCDF})+/F3 = \boldsymbol{ABCDEF}$. Preserves $\boldsymbol{A \rightarrow EF}$
  $A \rightarrow B$ $(R1)$, $B \rightarrow F$ $(R2)$ implies $A \rightarrow F$
  $A \rightarrow D$ $(R1)$, $D \rightarrow E$ $(R3)$ implies $A \rightarrow E$

Dependency Preservation (7): Example

- $R(ABCDEF) : F = \{A \rightarrow BCD, A \rightarrow EF, BC \rightarrow AD, BC \rightarrow E, BC \rightarrow F, B \rightarrow F, D \rightarrow E\}$. $Decomp = \{ABCD, BF, DE\}$
- On projections:

ABCD (R1)	BF (R2)	DE (R3)
A → B, A → C, A → D, BC → A, BC → D	B → F	D → E

- Infer reverse FD's:
  - $B+/F = BF : B \rightarrow A$ cannot be inferred
  - $C+/F = C : C \rightarrow A$ cannot be inferred
  - $D+/F = DE : D \rightarrow A$ and $D \rightarrow BC$ cannot be inferred
  - $A+/F = ABCDEF : A \rightarrow BC$ can be inferred, but it is equal to $A \rightarrow B$ and $A \rightarrow C$
  - $F+/F = F : F \rightarrow B$ cannot be inferred
  - $E+/F = E : E \rightarrow D$ cannot be inferred
- Need to check for: $\cancel{A \rightarrow BCD}, \boldsymbol{A \rightarrow EF}, \cancel{BC \rightarrow AD}, \boldsymbol{BC \rightarrow E}, \boldsymbol{BC \rightarrow F}, \cancel{B \rightarrow F}, \cancel{D \rightarrow E}$
  - $\boldsymbol{(BC)+/F = ABCDEF}$. Preserves $\boldsymbol{BC \rightarrow E, BC \rightarrow F}$
  - $\boldsymbol{(A)+/F = ABCDEF}$. Preserves $\boldsymbol{A \rightarrow EF}$

Practice Problems on Dependency Preservation

- **Check whether the decomposition of R into D is preserving dependency:**

a) $R(ABCD) : F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$. $D = \{AB, BC, CD\}$
b) $R(ABCDEF) : F = \{AB \rightarrow CD, C \rightarrow D, D \rightarrow E, E \rightarrow F\}$. $D = \{AB, CDE, EF\}$
c) $R(ABCDEG) : F = \{AB \rightarrow C, AC \rightarrow B, BC \rightarrow A, AD \rightarrow E, B \rightarrow D, E \rightarrow G\}$. $D = \{ABC, ACDE, ADG\}$
d) $R(ABCD) : F = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$. $D = \{AB, BC, BD\}$
e) $R(ABCDE) : F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$. $D = \{ABCE, BD\}$

Module Summary

- Understood the Characterization for and Determination of Lossless Join
- Understood the Characterization for and Determination of Dependency Preservation

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.
Edited and new slides are marked with "PPD".