Week 5 Lecture 1

Class	BSCCS2001
Created	@October 4, 2021 11:55 AM
Materials	
Module #	21
Type	Lecture
Week #	5

Relational Database Design

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 over time
- 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 ඞ

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

Database Management Systems Partha Pratim Das

- Consider combining relations
    - *sec_class(sec_id, building, room_number)* and
    - *section(course_id, sec_id, semester, year)*
- No repetition in this case

Redundancy and Anomaly

- **Redundancy:** Having multiple copies of the same data in the DB
    - This problem arises when a DB 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-normalized DBs where all the data is stored in one table (a flat-file DB)

  There can be 3 kinds of anomalies
    - Insertions anomaly
    - Deletion anomaly
    - Update anomaly
- **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 a large number of Instructors in *instructor_with_department* application may miss some of them

- 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, honor the dependencies, be efficient, etc
    - Various schemes of normalization ensure good decomposition
    - **Normalization** $\Rightarrow$ **Good decomposition**

Decomposition

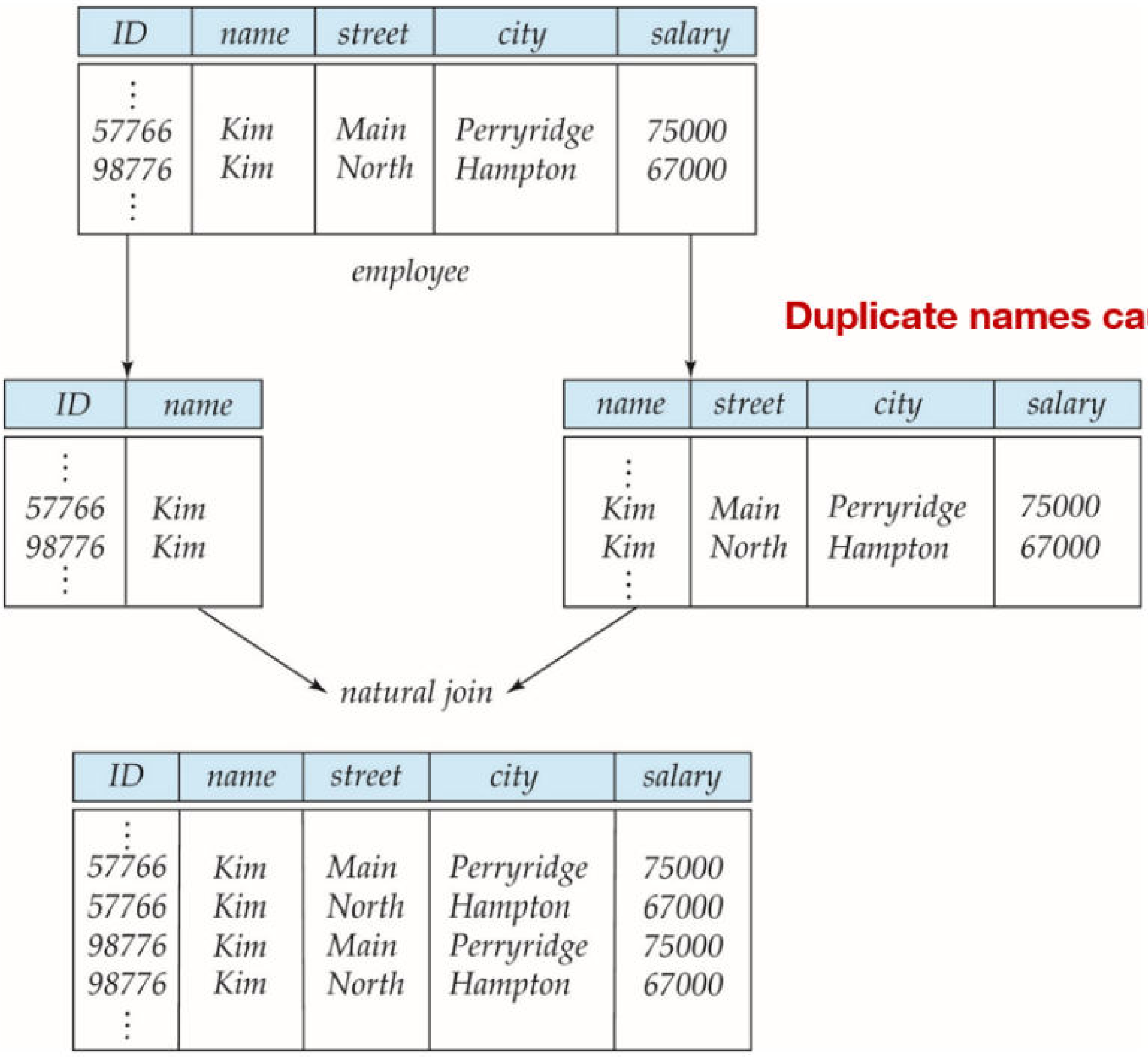
- Suppose we had started with *inst_dept*
- How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
- Denote as a **functional dependency**: *dept_name* $\rightarrow$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

  employee(<u>ID</u>, name, street, city, salary) into

    employee1 (<u>ID</u>, name)

    employee2 (name, street, city, salary)
- Note that if *name* can be duplicate, then *employee2* 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: Lossy Decomposition



Duplicate names cause the problem here.

Decomposition: 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

- **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 the redundancy from DBs 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 (1NF)** 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 the first normal form***
- **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 the roll numbers is not atomic
  - *Doing so is a bad idea ...*
    - It leads to encoding of the information in application program rather than in the database
- 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

- 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 the attributes

        - How to search telephone numbers
        - Why only two numbers?

- 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)

- 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

Week 5 Lecture 2

Class	BSCCS2001
Created	@October 4, 2021 2:41 PM
Materials	
Module #	22
Type	Lecture
Week #	5

Relational Database Design (part 2)

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 $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- The theory is based on:
  - Functional dependencies
  - Multi-valued dependencies
  - Other dependencies

Functional Dependencies

- Constraints on the set of legal relations
- Require that the values 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

---

- 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 on r agree on the attributes α, they also agree on the attributes β

  That is:

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

- **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

---

- 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 id
  - $K \rightarrow R$ and
  - for no $\alpha \subset K, \alpha \rightarrow R$
- Functional dependencies allows us to express constraints that cannot be expressed using superkeys
- Consider the schema:

  inst_dept(<u>ID</u>, name, salary, <u>dept_name</u>, 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*

- 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

- A functional dependency is trivial if it is satisfied by all instance of the relation
  - Example:
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*

- In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

- 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 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: 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 infinite 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: Closure of a Set of FDs

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

Week 5 Lecture 3

Class	BSCCS2001
Created	@October 4, 2021 3:28 PM
Materials	
Module #	23
Type	Lecture
Week #	5

Relational Database Design (part 3)

Functional Dependency Theory

Functional Dependencies: 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: 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 continued 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: 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: 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: Closure of Attribute Sets: Example

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^+$
  - result = AG
  - result = ABCG ($A \rightarrow C$ and $A \rightarrow B$)
  - result = ABCGHI ($CG \rightarrow H$ and $CG \rightarrow AGBC$)
  - result = ABCGHI ($CG \rightarrow I$ and $CG \rightarrow AGBCH$)
- Is AG a candidate key?

- Is AG a super key?
  - Does $AG \to R$? == Is $(AG)^+ \supseteq R$
- Is any subset of AG a superkey?
  - Does $A \to R$? == Is $(A)^+ \supseteq R$
  - Does $G \to R$? == Is $(A)^+ \supseteq R$

Functional Dependencies: Closure of Attribute Sets: Use

There are several uses of the attributes closure algorithm:

- Testing for superkey:
  - To test is 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 $\alpha \to \beta$ 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 $\gamma \to S$

Decomposition using Functional Dependency

BCNF: Boyce-Codd Normal Form

- A relations schema R is in BCNF w.r.t a set F of FDs if for all FDs in F^+ of the form

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

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

BCNF: Decomposition

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

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

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 if lossless join decomposition using FD set, the 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)
    $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$
- Prove that BCNF ensures Lossless Join

BCNF: 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: $R_1 = ZC, R_2 = CSZ - (C - Z) = SZ$
  - $R_1 \cup R_2 = CSZ = R, R_1 \cap R_2 = Z = \phi$ 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
    (**NOTE:** 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 must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation

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 3 potential problems to consider:

- May be impossible to re-construct 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 DB 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

  - 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)
- 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)

Week 5 Lecture 4

Class	BSCCS2001
Created	@October 4, 2021 6:10 PM
Materials	
Module #	24
Type	Lecture
Week #	5

Relational Database Design (part 4)

Algorithms for Functional Dependencies

Attribute Set Closure

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

Attribute Set Closure: Uses

There are several uses of the attribute closure algorithm

- Testing for a superkey:
  - To test if α is a superkey, we compute α^+ and check if α^+ contains all the 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
  - It 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: 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
  - Compute $(\{\alpha\} - A)^+$ using the dependencies in F
  - 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
  - Compute α^+ using only the dependencies in $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - 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 $F^+ = G^+$
  - That is: $(F^+ = G^+) \Leftrightarrow (F^+ \Rightarrow G \ and \ G^+ \Rightarrow F)$
  - Equivalence means that every functional dependency in F can be inferred from G and every functional dependency in G can 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 numbers 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 to work with?
- A **Canonical Cover** for F is a set of dependencies F_c such that ALL the following properties are satisfied:
  - $F^+ = F_c^+$
    - F logically implies all dependencies in F_c
    - F_c logically implies all dependencies in F
  - No functional dependency in F_c contains an irrelevant 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: 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: $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$

– 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: 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: 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

- 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

  irrelevant attribute either in α or in β

  /* NOTE: test for irrelevant attributes done using F_c, not F */

  If an irrelevant attribute is found, delete it from $\alpha \rightarrow \beta$

  until F does not change

- **NOTE:** Union rule may become applicable after some irrelevant attributes have been deleted, so it has to be re-applied

Canonical Cover: Example

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

Practice Problems on Functional Dependencies

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

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

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

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

> - **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 $X \to A$, then, X is also Candidate Key.
> - **Non Prime Attribute**: Attribute set does not belong to any candidate key are called Non Prime Attributes

- **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) $F : A \to C, AC \to D, E \to AD, E \to H$
     ii) $G : A \to CD, E \to AH$
  b) Consider the two sets P and Q with their FDs as below :
     i) $P : A \to B, AB \to C, D \to ACE$
     ii) $Q : A \to BC, D \to AE$

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

Week 5 Lecture 5

Class	BSCCS2001
Created	@October 4, 2021 8:05 PM
Materials	
Module #	25
Type	Lecture
Week #	5

Relational Database Design (part 5)

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: 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 (<u>S#</u>, Sname, City, Qty): Parts (<u>P#, Qty</u>)**

- Take natural join to reconstruct: **Supplier ⋈ 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#)** $\rightarrow$ **Qty**

- 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, Qty): Parts (P#, Qty)**
- Take natural join to reconstruct: **Supplier ⋈ 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 the original relation. **Join is lossless.**
- Common attribute **S#** is a superkey in **Supplier**
- Preserve all the dependencies

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - 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 Problem on Lossless join

- **Check if the decomposition of R into D is lossless:**
    a) $R(ABC) : F = \{A \to B, A \to C\}$. $D = R_1(AB), R_2(BC)$
    b) $R(ABCDEF) : F = \{A \to B, B \to C, C \to D, E \to F\}$.
    $D = R_1(AB), R_2(BCD), R_3(DEF)$
    c) $R(ABCDEF) : F = \{A \to B, C \to DE, AC \to F\}$. $D = R_1(BE), R_2(ACDEF)$
    d) $R(ABCDEG) : F = \{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$
        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) $R(ABCDEFGHIJ) : F = \{AB \to C, B \to F, D \to IJ, A \to DE, F \to GH\}$
        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

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

      $(F_1 \cup F_2 \cup ... \cup F_n)^+ = F^+$
    - If it is not, then checking updates for violation of functional dependencies may require computing join, 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: Testing

- To check if a dependency $\alpha \to \beta$ is preserved in a decomposition of R into $D = \{R_1, R_2, ..., R_n\}$ 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 $(F_1 \cup F_2 \cup ... \cup F_n)^+$

Dependency Preservation: Example

- $R\ (A, B, C, D, E, F)$

$F = \{A \to BCD, A \to EF, BC \to AD, BC \to E, BC \to F, B \to F, D \to E\}$

- Decomposition: $R1(A, B, C, D)$ $R2(B, F)$ $R3(D, E)$
  - $A \to BCD$, $BC \to AD$ are preserved on table R1
  - $B \to F$ is preserved on table R2
  - $D \to E$ is preserved on table R3
  - We have to check whether the remaining FDs: $A \to E$, $A \to F$, $BC \to E$, $BC \to F$ are preserved or not

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

- $F' = F_1 \cup F_2 \cup F_3$
- Checking for: $A \to E$, $A \to F$ in F'^+
  - $A \to D$ (from R1), $D \to E$ (from R3) : $A \to E$ (By transitivity)
  - $A \to B$ (from R1), $B \to F$ (from R2) : $A \to F$ (By transitivity)
- Checking for: $BC \to E$, $BC \to F$ in F'^+
  - $BC \to D$ (from R1), $D \to E$ (from R3) : $BC \to E$ (by transitivity)
  - $B \to F$ (from R2) : $BC \to F$ (by augmentation)

- $R(A, B, C, D)$

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

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

- $F' = F_1 \cup F_2 \cup F_3$
- Checking for: $D \to A$ in F'^+
  - $D \to C$ (from R3), $C \to B$ (from R2), $B \to A$ (from R1) : $D \to A$ (by transitivity)

    Hence, all dependencies are preserved

Dependency Preservation: Testing

- To check if a dependency $\alpha \to \beta$ is preserved in a decomposition of R into $R_1, R_2, ..., R_n$, we apply the following test (with attribute closure done with respect to F)
  - result = α

    while (changes to result) do

    for each R_i in the decomposition

    t = $(result \cap R_i)^+ \cap R_i$

    result = result ∪ t
  - If result contains all attributes in β, then the functional dependency $\alpha \to \beta$ is preserved
- We apply the test of 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 $(F_1 \cup F_2 \cup ... \cup F_n)^+$

Dependency Preservation: Example

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

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

- Need to check for: $\cancel{A \to BCD}$, $\boldsymbol{A \to EF}$, $\cancel{BC \to AD}$, $\boldsymbol{BC \to E}$, $\boldsymbol{BC \to F}$, $\cancel{B \to F}$, $\cancel{D \to E}$
- $(BC)+/F1 = ABCD$. $(ABCD)+/F2 = ABCDF$. $(\boldsymbol{ABCDF})+/\boldsymbol{F}3 = \boldsymbol{ABCDEF}$. Preserves $\boldsymbol{BC \to E}, \boldsymbol{BC \to F}$
  $BC \to AD$ $(R1)$, $AD \to E$ $(R3)$ implies $BC \to E$
  $B \to F$ $(R2)$ implies $BC \to F$
- $(A)+/F1 = ABCD$. $(ABCD)+/F2 = ABCDF$. $(\boldsymbol{ABCDF})+/\boldsymbol{F}3 = \boldsymbol{ABCDEF}$. Preserves $\boldsymbol{A \to EF}$
  $A \to B$ $(R1)$, $B \to F$ $(R2)$ implies $A \to F$
  $A \to D$ $(R1)$, $D \to E$ $(R3)$ implies $A \to E$

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