Lecture 13: Design Theory: Functional Dependencies and Normal Forms, Part II

Instructor: Shel Finkelstein

Reference:
A First Course in Database Systems,
3rd edition, Chapter 3, Sections 3.1 - 3.5

Important Notices

- Lecture slides and other class information will be posted on Piazza (not Canvas).
  - Slides are posted under Resources→Lectures
  - Lecture Capture recordings are available to all students under Yuja.
    - That includes classes given over Zoom.
  - There's no Lecture Capture for Lab Sections.
- All Lab Sections (and Office Hours) normally meet In-Person, with rare exceptions announced on Piazza.
  - Some slides for Lab Sections have been posted on Piazza under Resources→Lab Section Notes
- Office Hours for TAs and me are posted in Syllabus and Lecture1, as well as in Piazza notice @7; that post also now includes hours for Group Tutors.
- Some suggestions about use (and non-use) of Generative AI systems such as ChatGPT were posted on Piazza in @41, with specific discussion about the Movies tables and the four Avatar queries in Lecture 4.

Important Notices

- The Fifth Gradiance Assignment, CSE 180 Winter 2024 #5, on Design Theory, was assigned on Friday, March 8.
  - It is due by 11:59pm on **Friday, March 15**, the last day of class.
- Explanation of two difficult Gradiance questions on transactions was posted on Piazza on Sunday, March 2 in @122.
- Answers for two of the "Practice" Relational Algebra queries (in magenta) were posted on Piazza on Sunday, March 2 in @126.
- Winter 2024 Student Experience of Teaching Surveys - SETs opened on Monday, March 4.
  - SETs close on Sunday March 17 at 11:59pm.
  - Instructors are not able to identify individual responses.
  - Constructive responses help improve future courses.

Important Notices

- The subject of Lab4 is Real Application Programming, using libpq (this Lecture) and a Stored Function (previous Lecture).
  - Lab4 was posted on Piazza on Wednesday, February 28.
  - Lab4 is hard, and many students will need help completing it.
    - Read the Lab4 announcement on Piazza as soon as possible!
    - We provided information files and examples, posted under Resources→Lab4, and described in the Lab4 pdf and announcement.
    - We have also provided load data that you can use to test your Lab4 solution.
      - But testing can prove that program is incorrect, not that it's correct.
  - Lab4 is due on **Tuesday, March 12 by 11:59pm**.
    - unix.ucsc.edu gets busy near the end of the quarter.
    - Please avoid Infinite Loops in your Stored Function.
  - All material needed for Lab4 was covered in Lecture by Friday, March 1.
  - However, there were some important clarifications and corrections which were made to Lab4.
    - These corrections, which are described in @121 on Piazza, "Important updates of multiple Lab4 files", are summarized on the next slide.

Important Updates of Multiple Lab4 Files Slide #1 (see @121)

- The Lab4 pdf is now correctly called CSE180_W24_Lab4.pdf
- A new version of the load file, load_lab4.sql, has been posted.
- create_lab4.sql fix: Every theaterID in productions must appear in Theaters.
- Several changes have been made in the runShakespeareApplication.c skeleton file.
  - The signature for the C function IncreaseSomeCastMemberSalaries in the runShakespeareApplication.c skeleton has been changed to:

    float increaseSomeCastMemberSalaries(PGconn *conn, char *thePlayTitle,
    int theProductionNum, float maxDailyCost)
    - That is, the maxDailyCost parameter is float, and the function returns a float value.
  - Minor: The name of the function begins with a lowercase ”i”, just as all the other C functions begin with lowercase letters.
    - Same is true for the Stored Function which is invokes, which is called increaseSomeCastMemberSalariesFunction (lowercase i, not uppercase).
  - Minor: Parameter which had been called theProdNum is now called theProductionNum, aligning with productionNum attribute name in tables.

Important Updates of Multiple Lab4 Files Slide #2 (see @121)

- The first lines of the Stored Function increaseSomeCastMemberSalariesFunction (which we did not give you in the Lab4 pdf) should be:

```
CREATE OR REPLACE FUNCTION
    increaseSomeCastMemberSalariesFunction(thePlayTitle VARCHAR(40),
        theProductionNum INTEGER, maxDailyCost NUMERIC(7,2))
RETURNS NUMERIC(7,2) AS $$
```

  - Note that C does not have a type corresponding to NUMERIC(7,2).
- The value returned by increaseSomeCastMemberSalariesFunction when you execute that Stored Function in your C program (and then use PQgetvalue(res,0,0) to get get the result of the Stored Function) is a string.
  - Just as you can convert an appropriate character string to an integer using C's atoi function, you can convert a character string to a float using C's atof function, and increaseSomeCastMemberSalaries needs to return a float.
- The value that you print out in your tests of increaseSomeCastMemberSalaries should be a number with 2 decimal places. You can print out a floating point value in that format using the %7.2f format string (instead of using %f or %d).
- Per @135, you should assume that salaryPerDay, theaterFeePerDay (and also theaterID in Productions) can't be NULL.

Important Notices: Final

- CSE 180 Final is on **Wednesday, March 20, 8:00-11:00am, and it will be given in-person in our classroom, except for students who receive Remote Exam permission.**
  - **No** early/late Exams. **No** make-up Exams. **No** devices (except for Remote students).
  - 3 hours, extended for DRC students, covering the entire quarter.
    - All DRC students should have recently received email about the final.
    - Final will be harder than the Midterm.
  - **You may bring one double-sided 8.5 x 11 sheet to the Final, with anything that you want written or printed on it that you can read unassisted) …**
    - (Okay, you may bring two sheets with writing only on one side of each sheet.)
    - **But you must not use any other material or receive any help during the Final, whether you're in the classroom, remote, or in a DRC room**
    - As the Syllabus emphasizes, Academic Integrity violations have serious consequences!
  - We will assign seats as you enter the room. You may not choose your own seat.
- CSE 180 Final from Winter 2023 was posted on Piazza under Resources→Exams on Sunday, March 3.
  - Solution to that Final was posted on Piazza on Sunday, March 10 … but take it yourself first, rather than just reading the solution.

Important Notices: In-Person Final

Final includes a Multiple Choice Section and a Longer Answers Section.

- In-Person students should bring a Red Scantron sheet (ParSCORE form number f-1712) sold at Bookstore for about 25 cents, and #2 pencils for Multiple Choice Section.
- You'll answer Multiple Choice Section on your Scantron sheet, entering your name, your student ID, and which version ("Test Form Letter") of the Multiple Choice Section you're answering.
  - Write your Multiple Choice answers on the Scantron sheet using a #2 pencil.
  - Ink and #3 pencils don't work.
- You'll answer Longer Answers Section on the Long Answers Section, using either ink or #2 pencil (as on the Midterm).
- Be sure to answer all questions readably!!
- Do not hand in your 8.5 x 11 sheet or your Multiple Choice Section. Just hand in your Scantron Sheet and your Longer Answers Section.
  - But show us your Multiple Choice Section, so that we can check that you filled in the Test Form Letter (A, B, C or D) for that Section on your Scantron Sheet.

Important Notices: Remote Final

CSE 180 Final **will be given in-person in our classroom, except for students who receive Remote Exam permission.**

- If you need to take the Final Remotely, send me an email whose subject is "Taking the CSE 180 Final Remotely" by Tuesday, March 19 at 6:00pm that includes your strong justification for that request.
  - Only students whose requests are approved may take the Final Remotely.
    - If you don't receive a response from me by 9:00pm, please send email again!
- I'll send instructions to you for taking the Final before 8:00am on Wednesday, March 20 which include a Zoom link
  - You must be on Zoom while you are taking the Final.
    - Please make sure that your face is visible on Zoom video, but that your microphone is muted.
    - Please do not have headphones on during the Final.
    - If there are any bugs on the Final, they will be conveyed to all Remote students via Zoom Chat.
    - If you have a question during the Final, please send it to me (just to me) directly using Zoom Chat, not by speaking.
  - You'll have to complete the exam by 11:00am, just like all other students …
  - … except for DRC students, who will receive extra time, whether they take the exam In-Person (in a different room) or Remotely.
  - You won't need a red Scantron Sheet if you're taking the Final Exam Remotely.

A Word to the Unwise

- This is a tough class for some students since it involves a combination of theory and practice.
  - The second half of CSE 180 is much harder than the first half of the course.
- Students who regularly attend Lectures and Lab Sections (and Office Hours and Tutoring) often do well; students who don't regularly attend often do poorly.
  - We don't take attendance; you're responsible for your own choices.
- After the course ends, your course grade will be determined by your scores on Exams, Labs and Gradiance, as described on the "Course Evaluation" and "Grading" Slides in the Syllabus.
  - You won't be able to do any additional work to improve your grade.

Functional-Dependency Theory Roadmap

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We'll define Normal Forms that may help us avoid Redundancy and Anomalies.
- Then we'll define Decompositions of relations into multiple relations (similar to our Employee/Rank example), and we'll explain what it means for a decomposition to be "Lossless".
- Textbook includes algorithms to decompose relations losslessly into "Normal Forms".
  - We'll discuss one simple approach.
- But we will briefly mention what it means for a decomposition to be Dependency-Preserving.
- And we'll end by explaining that 2 out of 3 ain't bad.
  - We'd like 3 nice properties, but we might only be able to get 2.
  - Anomaly avoidance, Lossless Decomposition, Dependency-Preservation

Normal Forms

Given a relation schema, we want to understand whether it is a good design or a bad design.

- Intuitively, a good design is one that does not store data redundantly, and does not lead to anomalies.

If we know that rank determines salary_scale, which is a better design?

Employees(eid, name, addr, rank, salary_scale)

OR

Employees2(eid, name, addr, rank)
Salary_Table(rank, salary_scale)

Remember that sometimes database designers **may choose** to live with redundancy in order to improve query performance. But then they'll have to cope with anomalies, which can be difficult.

First Normal Form (1NF)

- A Relation Schema is in *first normal form (1NF)* if the type of every attribute is atomic.
- Very basic requirement of the relational data model.
  - Not based on FDs.
  - **Every other Normal Form that we'll discuss assumes 1NF, even though we won't bother saying that.**

Example:

R(ssn: char(9), name: string, age: int)

- All our examples so far have been in 1NF.

Example of a non-first normal form relation:

R(ssn: char(9), name: Record[firstname: string, lastname: string], age: int, children: Set(string))

Second Normal Form (2NF)

- Not particularly important
  - We won't discuss this.
  - (Neither does the textbook.)

Keeping FDs Simple

- We proved the following in the previous Lecture:

  Union Rule:

  If X → Y and X → Z, then X → YZ

  Decomposition Rule:

  If X → YZ, then X → Y and X → Z

- So from now on, we'll assume that right side of every FD is a single attribute.
  - For example, instead of writing AC → BDE, we will write:

    AC → B, AC → D, AC → E

Boyce-Codd Normal Form (BCNF)

- Let R be a relation schema, $\mathcal{F}$ be a set of FDs that holds for R, with A as an attribute in R, and X as a subset of the attributes in R.

- R is in *Boyce-Codd Normal Form (BCNF)* if
  - For every FD X → A in $\mathcal{F}$, at least one of following is true:
    - X → A is a trivial FD (i.e., A ∈ X) or,
    - X is a superkey for R.

- BNCF is desirable for avoiding redundancy.
  - Recall our Employees2/Salary_Table example.

Is this relation in BCNF?

A	B	C
a1	b1	c1
a1	b2	c1

- The only functional dependency given is A → C.
- (to fill in)

Is this relation in BCNF?

A	B	C
a1	b1	c1
a1	b2	c1

- The relation is <u>not</u> in BCNF because:

  A → C is not a trivial FD and A is not a superkey.

- Given that A → C, we can infer that C value of second tuple is also c1.
- But note that c1 is redundantly stored twice.

Keys and Prime Attributes

- Supposed that DriversLicenses(name, state, licenseNum, ssn) is a relation schema in which the non-trivial FDs are:

  (state, licenseNum) → name

  (state, licenseNum) → ssn

  ssn → name

  ssn → state

  ssn → licenseNum

  Then both {state, licenseNum} and {ssn} are keys.

  - Why are they keys?
  - Is {state, licenseNum, ssn} a key?

Convention: We will often omit the set brackets, and say that state, licenseNum is a key and ssn is a key.

- If an attribute A is part of some key of relation schema R, then A is is referred to as a Prime Attribute of R.
  - In the above example, each of the attributes state, licenseNum and ssn is a Prime Attribute. But name isn't a Prime Attribute. (Why not?)

Third Normal Form (3NF)

- Let R be a relation schema, $\mathcal{F}$ be a set of FDs that holds for R, with A as an attribute in R, and X as a subset of the attributes in R.

- R is in *Third Normal Form (3NF)* if
  - For every FD X → A in $\mathcal{F}$, at least one of following is true:
    - X → A is a trivial FD (i.e., A ∈ X), or
    - X is a superkey for R, or
    - A is a Prime Attribute.
      That is, A is part of some key of R.

- Note that the red condition says that A is part of some key for R, not part of some superkey for R.

BCNF/3NF Example 1

Consider R(A, B, C, D)

with non-trivial FDs: ABC → D, A → D

Trivial FDs are implicit. They are not listed because they always hold, and BCNF covers them.

- Is R in BCNF?
- Is R in 3NF?

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

BCNF/3NF Example 2

Now consider R(A, B, C, D)

with FD's: ABC → D, A → D, D → A.

– Is ABC a key for R? Is BCD a key for R?

- Is R in BCNF?
- Is R in 3NF?

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

- Note that there is still redundancy in R, even though it is in 3NF!

BCNF and 3NF

- By definition, any BCNF relation must also be a 3NF relation.

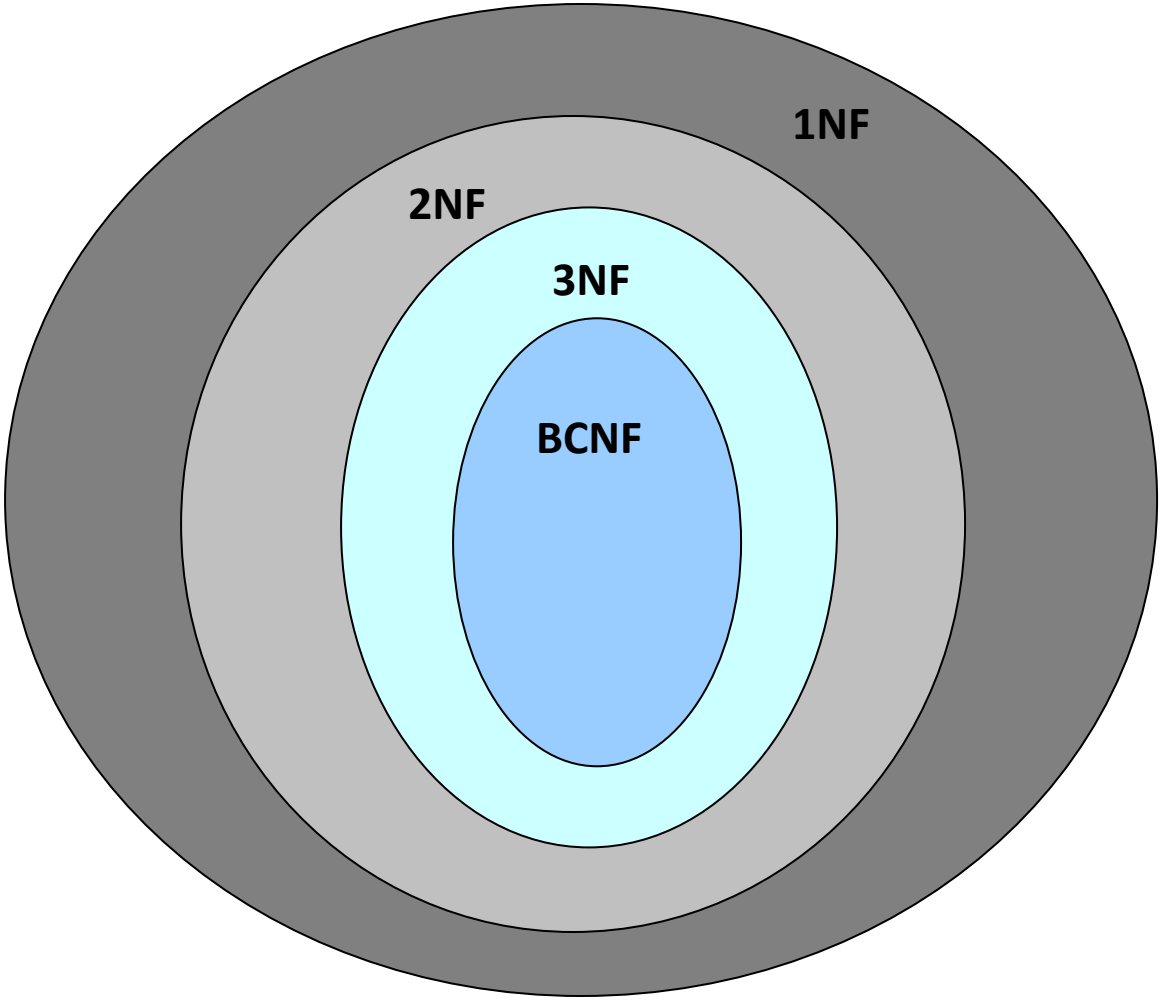
- Definition says:

  … if at least one of the following holds for each FD X → A:

  - X → A is a trivial dependency (i.e., A ∈ X). BCNF, 3NF
  - X is a superkey for R. BCNF, 3NF
  - A is a Prime Attribute for R. 3NF

- However, a 3NF relation is not always in BCNF.
  - Example 2 is an example of a 3NF relation that is **not** in BCNF.

Relationships Among Normal Forms – The Big Picture



BCNF/3NF Example 3

Company_Info(emp, dept, manager)

emp → dept, emp → manager, dept → manager

Is it in BCNF?

Is it in 3NF?

BCNF/3NF Example 4

PostOffice(city, street, zip)

city, street → zip

zip → city

The above FDs are true of most post office policies. Note that a city may have multiple zips, but a zip must be in a single city.

Is PostOffice in BCNF?

Is PostOffice in 3NF?

- Despite 3NF, there can be Redundancy: The association of a zip with a city could appear in multiple records of PostOffice.
- **So although** PostOffice **is in 3NF, there can be Anomalies:**
  - zip → city. So if the city was changed in one (city, street, zip) record, but was not changed for another (city, street, zip) record that has the same zip, that would be an anomaly.
  - Example: (Santa Cruz, Heller St, 95064) and (Santa Cruz, McLaughlin Dr, 95064)

BCNF/3NF Example 5

Customers(ssn, name, address)

ssn → name

ssn → address

Is Customers in BCNF?

Is Customers in 3NF?

Algorithm for Testing Whether a Relation is in BCNF using Attribute Closure

Given R and $\mathcal{F}$, determine whether R is in BCNF.

- For each FD X→Y ∈ $\mathcal{F}$ such that Y ⊈ X (i.e., the FD is non-trivial), compute X^+.
  - If every such X is a superkey (i.e., X^+ = attr(R)), then R is in BCNF.
  - If there is a set X of attributes such that $X^+ \neq$ attr(R), then R is not in BCNF.

Examples: BCNF Testing

- CompanyInfo(emp, dept, manager)
  - emp → dept, dept → manager
  - $dept^+ \neq$ attr(CompanyInfo). Hence CompanyInfo **is not** in BCNF.

- Customers(ssn, name, address)
  - ssn → name
  - ssn → address
  - ssn^+ = attr(Customers) Hence Customers **is** in BCNF.

- PostOffice(city, street, zip)
  - city, street → zip
  - zip → city
  - $zip^+ \neq$ attr(R) Hence PostOffice **is not** in BCNF.

Binary Relations and BCNF

Is R(A,B) in BCNF?

- We haven't told you the Functional Dependencies on R(A,B)!
  - Consider all the possibilities,

- **Fact**: Any binary relation schema is in BCNF.
  - Why?

“Improving” a non-BCNF Relation

How can we “improve” a relation R that is not in BCNF?

- **Approach**: Decompose (“break up”) relation R into smaller relations, so that each smaller relation is in BCNF.
- We did this when we decomposed Employees, separating out Salary_Table because of the FD: rank → salary_scale

  Employees2(<u>eid</u>, name, addr, rank)

  Salary_Table(<u>rank</u>, salary_scale)

Decomposition of a Relation

A *decomposition of a relation R* is defined by sets of attributes $X_1, \ldots, X_k$ (which don't have to be disjoint) such that:

1. Each $X_i \subseteq$ attr(R)
2. $X_1 \cup X_2 \cup \ldots \cup X_k$ = attr(R)

For a decomposition, we will write $\pi_{Xi}(R)$ as R_i, with instances of R written as r and instances of R_i written as r_i.

Examples:

- CompanyInfo(emp, dept, manager)
  - R_1(emp, dept), R_2(dept, manager)
- R(A,B,C,D,E,F,G)
  - R_1(A,C), R_2(A,B,C,D), R_3(C,D,E,F,G)

Goals for Redesigning Schema Using A Decomposition

1. The decomposition Eliminates Anomalies.
2. The decomposition doesn't lead to any "extra data" (that was not in instance r) when the r_i's are re-joined back together.
   - Such Decompositions are called Lossless Join decompositions.
   - Why must the Natural Join of all the r_i's always give at least all the data that was in r?
3. Dependency Preservation: (not required for Final)
   - The FD's on R_i are the FD's in $\mathcal{F}^+$ that mention only attr(R_i).
   - The decomposition is Dependency-Preserving if when the R_i's are re-joined back together, the FD's that were on the R_i's imply all of the original FD's in $\mathcal{F}$.

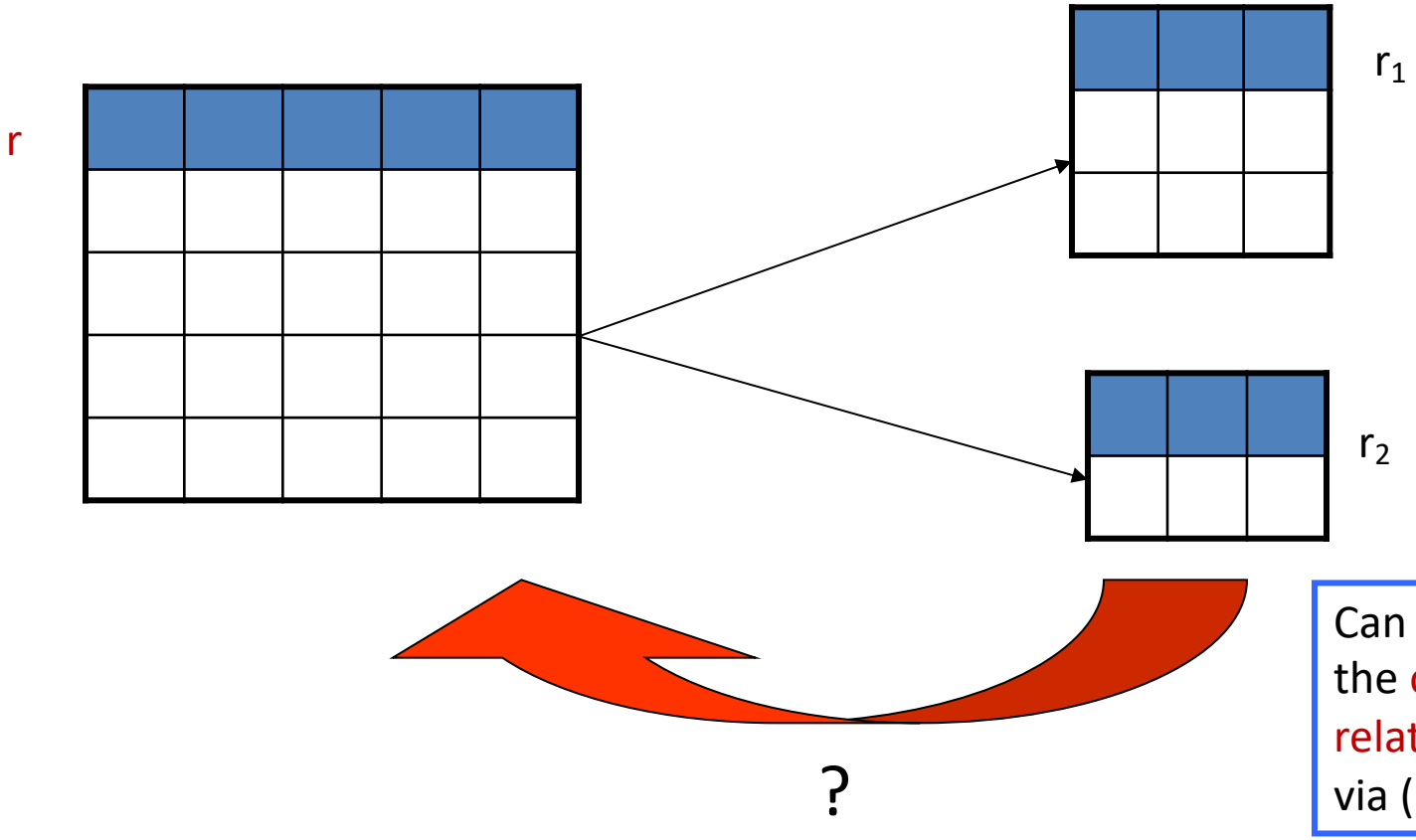
Are All BCNF Decompositions "Good"?

- Is it always possible to decompose R so that each smaller relation is in BCNF?
  - YES
  - One strategy: decompose R into a set of relation schemas R_1, …, R_k such that each R_i is a binary relation schema.

- Are all BCNF decompositions "good"?
  - NO

Decomposing a Relation

- Suppose we have decomposed R into R_1 and R_2. Given an instance r of R, we decompose r into r_1 and r_2. Can we get back the original instance r by (Natural) Joining r_1 and r_2?



Can we reconstruct the original relation instance r via (Natural) Join?

Lossless Join Decomposition

In general, can we obtain r by Natural Joining r_1 with r_2 … with r_k?

- That is, must it always true for any instance r, that:
  $r = r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$ (Natural Join) ?

Parentheses not shown
Because ⋈ is associative.

More precise definition:

- Let R be a relation schema and $\mathcal{F}$ be a set of FDs over R.
- A decomposition of R into *k* schemas, with attribute sets X_1 , …, X_k, is a *Lossless Join decomposition with respect to* $\mathcal{F}$ if:

For every instance r of R that satisfies $\mathcal{F}$, we have:

$$r = \pi_{X1}(r) \bowtie \ldots \bowtie \pi_{Xk}(r)$$
$$= r_1 \bowtie r_2 \bowtie \ldots \bowtie r_k$$

Lossless Join Example 1

- Let R(A,B,C) be a relation schema with no functional dependencies
- Is the decomposition of R into schemas $R_1(A,B)$ and $R_2(B,C)$ a Lossless Join decomposition?

Instance rx

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c3

$\pi_{A, B}(rx)$

A	B
a1	b1
a1	b2

$\pi_{B,C}(rx)$

B	C
b1	c1
b1	c2
b2	c3

$\pi_{A, B}(rx) \bowtie \pi_{B,C}(rx)$

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c3

Lossless Join Example 2

Instance ry

A	B	C
a1	b1	c1
a2	b1	c2

$\pi_{A, B}(ry)$

A	B
a1	b1
a2	b1

$\pi_{B,C}(ry)$

B	C
b1	c1
b1	c2

$\pi_{A, B}(ry) \bowtie \pi_{B,C}(ry)$

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c2

Lossy!

- By projecting on $R_1(A,B)$ and $R_2(B,C)$, some information may be lost (sometimes).
- We no longer know that (a1,b1,c2) did not exist in the original relation!
- Hence R_1 and R_2 is <u>not</u> a Lossless Join decomposition of R.

FD's and Lossless Joins

- Let R(A,B,C) be a relation schema
- Is the decomposition of R into schemas R_1(A,B) and R_2(B,C) a Lossless Join decomposition if we know B → C?
  - ry is not a legal instance, since it does not satisfy B → C.
- But that doesn't prove that R_1(A,B) and R_2(B,C) is a Lossless Join decomposition in the presence of the FD B → C.
  - Is it Lossless?
  - Yes; see textbook, Sections 3.4.1 and 3.4.2 …
  - … or later slides in this lecture!!

Lossless Join Example 3

CompanyInfo(emp, salary, dept, manager)

emp → salary, dept, manager

dept → manager

- CompanyInfo is not in BCNF because of dept → manager.
- Let's decompose into R_1(emp, salary) and R_2(dept, manager).

Instance r of CompanyInfo:

(Bolt, 85K, Math, Trombley)

(Montgomery, 90K, Math, Trombley)

(Brandt, 88K, CS, Pohl)

Lossless Join Example 3 (cont'd)

- Decompose instance r of CompanyInfo(emp, salary, dept, manager)
  - r_1(emp, salary)
    (Bolt, 85K)
    (Montgomery, 90K)
    (Brandt, 88K)
  - r_2(dept, manager)
    (Math, Trombley)
    (CS, Pohl)
- $r_1 \bowtie r_2 = r_1 \times r_2$ has 6 tuples in it.
- But instance r had only 3 tuples in it!
  - Hence the decomposition of CompanyInfo(emp, salary, dept, manager) into R_1(emp, salary) and R_2(dept, manager) is not a Lossless Join decomposition.

A Necessary and Sufficient Condition for Lossless Join Decomposition

- We would like our decompositions to be Lossless, and we'd like to be able to decide when a decomposition is Lossless.

Let R be a relation and $\mathcal{F}$ be set of FDs that hold over R.

Fact: A decomposition of R into two relation schemas R_1 with attributes X_1 and R_2 with attributes X_2 is Lossless if and only if $\mathcal{F}^+$ contains either:

1. $X_1 \cap X_2 \rightarrow X_1$, or
2. $X_1 \cap X_2 \rightarrow X_2$

That is, the intersection of the attributes of R_1 and R_2 is a superkey of either R_1 or R_2

Testing Whether a Decomposition is a Lossless Join Decomposition

Fact: A decomposition of R into relation schemas R_1 and R_2 is Lossless if and only if $\mathcal{F}^+$ contains either:

1. $X_1 \cap X_2 \rightarrow X_1$, or
2. $X_1 \cap X_2 \rightarrow X_2$

That is, the intersection of the attributes of R_1 and R_2 is a superkey of either R_1 or R_2

- This **Fact** works only for decompositions into **two** relations.
  - And note that it's not the definition of Lossless Join Decomposition!
- "The Chase" (see textbook) is a procedural algorithm for checking whether any decomposition is a Lossless Join decomposition.
  - We won't cover "The Chase" in this class.

Some Lossless Join Questions

- What if we decomposed R(A,B,C), with no non-trivial FDs (so $\mathcal{F}$, is the empty set) into R1(A,B) and R2(B,C),
  - Since B → AB and B → BC are not in $\mathcal{F}^+$, this decomposition is not a Lossless Join decomposition.
- CompanyInfo(emp, salary, dept, manager), with FDs:

  emp → salary, dept, manager

  dept → manager

  CompanyInfo is not in BCNF.
  - What if we decomposed CompanyInfo into R1(emp, salary) and R2(dept, manager)
    - Since FDs {} → emp,salary and {} → dept, manager are not in $\mathcal{F}^+$, this is not a Lossless Join decomposition.
  - But what if we decomposed CompanyInfo into: CInfo1(emp, salary, dept) and CInfo2(dept, manager)?
    - Does that work?

A Final Lossless Join Example

Employees(eid, name, addr, rank, salary_scale)
with FD: rank → salary_scale

eid abbreviates that eid is a **key** for Employees.

Decomposition:

Employees2(eid, name, addr, rank)
Salary_Table(rank, salary_scale)

attr(Employees2) ∩ attr(Salary_Table) = {rank}
rank → attr(Salary_Table).
Therefore, the decomposition is Lossless.

A Simple Approach to Convert a non-BCNF Schema into a BCNF Schema

1. Suppose that DB schema has a relation R that is not in BCNF.
   - Then there must be at least one non-trivial FD X → Y such that X is not a superkey for R.
   - We can assume that none of the attributes in X appear in Y.
     - (If some X attributes are in Y, we can just remove them from Y.)
2. Replace R with two relations which have the following attributes:
   - attr(R) – Y keeping the FDs on R which don't include any attributes of Y
   - X U Y with FD X → Y, as well as all other FDs on attributes X U Y
3. If resulting DB schema is still not in BCNF, go back to Step 1!

This is exactly what we did when we began the Design Theory lectures!

Employees(eid, name, addr, rank, salary_scale)
with FD: rank → salary_scale

was replaced by:

Employees2(eid, name, addr, rank)

Salary_Table(rank, salary_scale)

Decomposition and Normalization

[Not required for Final]

Given a relation schema and functional dependencies, it is always possible to decompose schema into a set of **BCNF** relations that:

- 1) *Eliminates Anomalies,*
- and is 2) a *Lossless Join* decomposition.
- However, the schema might not always be 3) *Dependency-Preserving.*

Given a relation schema and functional dependencies, it is always possible to decompose schema into a set of **3NF** relations that:

- is 2) a *Lossless Join* decomposition,
- and is *3) Dependency-Preserving*.
- However, the schema might not always 1) *Eliminate Anomalies.*