CSE 4020/5260
Database Systems

Instructor: Fitzroy Nembhard, Ph.D.

Week 12-13

Database Design

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY®

Distribution

- All slides included in this class are for the exclusive use of students and instructors associated with Database Systems (CSE 4020/5260) at the Florida Institute of Technology
- Redistribution of the slides is not permitted without the written consent of the author.

Relational Database Design

- Goals of Relational Database Design
- Functional Dependencies
- Loss-less Joins
- Dependency Preservation
- Normal Forms (1st, 2nd, 3rd, BCNF)

Reading: Chapter 7

Goals of Relational Database Design

- Traditional Design Goals:
  - Avoid redundant data – generally considered enemy #1.
  - Ensure that relationships among attributes are represented.
  - Facilitate the checking of updates for violation of integrity constraints.

- We will formalize these goals in several steps.

The Database Design Process

- Database design is driven by *normalization*.

- If relational scheme R is not sufficiently normalized, *decompose* it into a set of relational schemes $\{R_1, R_2, ..., R_n\}$ such that:
  - Each relational scheme is sufficiently *normalized*.
  - The decomposition has a *lossless-join*.
  - All functional dependencies are *preserved*.

- So, what are **normalization, lossless-join, functional dependencies**, and what does preserving them mean?

The Database Design Process – Notations and Definitions

- The notation $r(R)$ refers to the relation r with schema R. When we write $r(R)$, we thus refer both to the relation and its schema
- $r(R)$ may be generated by converting an E-R diagram to a set of relation schemas.
- $r(R)$ may be a single relation schema containing *all* attributes that are of interest. The normalization process then breaks up $r(R)$ into smaller schemas.
- $r(R)$ may be the result of an ad hoc design of relations that we then test to verify that it satisfies a desired normal form.
- For the majority of this set of slides, we assume that a relation schema $r(R)$ is given, and we will proceed to normalize it or check dependency preservation.
- A proper subset of a set A is a subset of A that is not equal to A. In other words, if B is a proper subset of A, then all elements of B are in A, but A contains at least one element that is not in B.
  - For example, if A={1,3,5} then B={1,5} is a proper subset of A, written as $B \subset A$. The set C={1,3,5} is a subset of A, written as $C \subseteq A$.

First Normal Form

- A domain is *atomic* if its elements are *treated* as indivisible units.
  - Examples of atomic domains:
    - Number of pets
    - Gender
  - Examples of non-atomic domains:
    - Person names
    - List of dependent first names
- A relational schema R is in *first normal form* if all attributes of R are atomic (or rather, are treated atomically).

Is the University Schema in 1st NF?

university schema

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

course

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

The Problem with Redundancy

- So why is redundancy considered “enemy #1?”
- Consider the relation schema:

Banking Schema
branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Note the redundancy in *branch-name, branch-city,* and *assets*.
  - Wastes space.
  - Creates insertion, deletion, and update anomalies.

Update, Insertion and Deletion Anomalies

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Insertion Anomalies:
  - Cannot store information about a branch if no loans exist without using null values; this is particularly bad since loan-number is part of the primary key.
  - Subsequent insertion of a loan for that same branch would require the first tuple to be deleted.
- Deletion Anomalies:
  - Deleting L-17 and L-14 might result in all Downtown branch information being deleted.
- Update Anomalies:
  - Modify the asset value for the branch of loan L-17.
  - Add $100 to the balance of all loans at a Brooklyn branch.

Banking Schema

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Decomposition

- **Solution** - decompose *Lending-schema* into:

  Branch-schema = (branch-name, branch-city, assets)

  Loan-info-schema = (customer-name, loan-number, branch-name, amount)

- For any decomposition:
  - All attributes of an original schema must appear in the decomposition*:*

    $$R = R_1 \cup R_2$$

  - The decomposition must have a *lossless-join*, i.e., for all possible relations *r* on schema *R:*

    $$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

Test Your Knowledge

- What if instead of instructor and department, I had a single table called:
  - *instructor_dept* (ID, name, salary, dept name, building, budget)?

classroom(*building*, *room_number*, *capacity*)
department(*dept_name*, *building*, *budget*)
course(*course_id*, *title*, *dept_name*, *credits*)
instructor(*ID*, *name*, *dept_name*, *salary*)
section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
student(*ID*, *name*, *dept_name*, *tot_cred*)
takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
advisor(*s_ID*, *i_ID*)
time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
prereq(*course_id*, *prereq_id*)

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

Example of Lossy-Join Decomposition

- Decomposition of $R = (A, B)$ into $R_1 = (A)$ and $R_2 = (B)$

A	B
α	1
α	2
β	1

r

A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_B(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B
α	1
α	2
β	1
β	2

Recall the Natural Join

- Relational schemes for relations r and s, respectively:

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

  $S = (E, B, D)$

- Resulting schema for $r \bowtie s$:

  (A, B, C, D, E)

  $\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

Example of Lossless Join Decomposition

- Decomposition of $R = (A, B, C)$ into $R_1 = (A, B)$ and $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

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

B	C
1	A
2	B

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

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Theory Connection: Lossy vs Lossless Compression

- Lossy compression refers to compression in which some of the data from the original file (JPEG) is lost
- Lossless compression refers to compression in which the image is reduced without any quality loss.

Source: https://www.keycdn.com/support/lossy-vs-lossless

Functional Dependencies

- Informally, a *Functional Dependency (FD)* is a constraint on the contents of a relation.

- An FD specifies that the values for one set of attributes determines the values for another set of attributes (in other words, FD is a relationship that exists when one attribute uniquely determines another attribute)
- The notion of an FD is a generalization of the notion of a *key* (super, candidate, primary or unique).
  - *In fact, in a "good" design, most FDs are realized as keys.*

Banking Schema
branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Functional Dependencies – Example #1

- Consider the following schema:

  Loan-info-schema = (customer-name, loan-number, branch-name, amount)

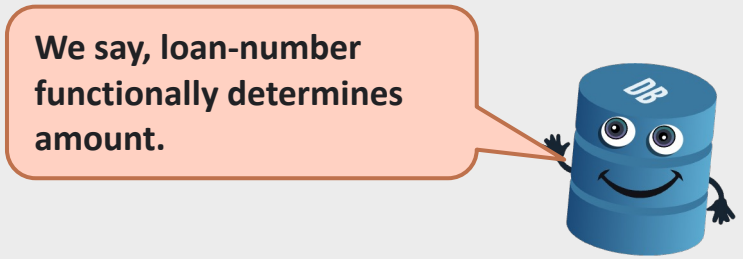
- Applicable FDs:

  loan-number → amount
  loan-number → branch-name

- Non-applicable FDs:

  loan-number → customer-name

  customer-name → amount



Functional Dependencies – Example #2

Student ID	Semester	Course	TA
901411123	Fall 2020	Database Systems	James
901411121	Fall 2021	Database Systems	Fred
901411123	Fall 2020	Operations Research	Bob
901411124	Summer 2021	Database Systems	Peter
901411124	Summer 2021	Physics II	Simon

StudentID → Semester

What about:
Course → TA
StudentID, Course → TA
StudentID, Course → TA, Semester

Is {StudentID, Course} a superkey?

Whenever two rows in this table feature the same *StudentID*, they also necessarily have the same *Semester* values

Functional Dependencies – Example #3

- Consider the following schema:

  Grade-schema = (student-id, name, course-id, grade)

- Applicable FDs:

  student-id $\rightarrow$ *name*

  student-id, course-id $\rightarrow$ *grade*

- Non-applicable FDs:

  student-id $\rightarrow$ *grade*

  grade $\rightarrow$ *course-id*

- Exercise – list out all possible FDs for the above relational scheme, and determine which ones hold and which ones don't (same for the one on the previous page).

Functional Dependency - Formal Definition

- Let *R* be a relation schema where $\alpha \subseteq R$ *and* $\beta \subseteq R$.

- The functional dependency

$$\alpha \rightarrow \beta$$

is said to hold 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, i.e.,

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

- Alternatively, if $\alpha \rightarrow \beta$ then the relation *r* can never contain two tuples that agree on α and disagree on β.

Trivial Functional Dependencies

- Let R be a relation schema where $\alpha \subseteq R$ *and* $\beta \subseteq R$.
- Some functional dependencies are said to be **trivial** because they are satisfied by all relations. For example, $A \rightarrow A$ is satisfied by all relations involving attribute A
- For all tuples $t1$ and $t2$ such that $t1[A] = t2[A]$, it is the case that $t1[A] = t2[A]$.
- Similarly, $AB \rightarrow A$ is satisfied by all relations involving attribute A. In general, a functional dependency of the form $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.

Use of Functional Dependencies

- Let R be a relational scheme and let F be an associated set of functional dependencies.
- *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F*
  - *F* is *imposed* or *enforced* on *R*.
- If a relation *r* is legal for a set *F* of functional dependencies, we say that *r* satisfies *F*
  - *F* is currently satisfied but may or may not be *imposed* or *enforced* on r.
- Note that the difference between the two is important!
  - If *F* holds on relation *R*, then every relation (i.e., a set of tuples) must satisfy *F*.
  - If a relation satisfies *F*, it may or may not be the case that *F* holds on *R*.

Example – FD Formal Definition

- An analogy - assume for the moment that all drivers actually follow speed limits...
- Thus, we say that the speed limit established for a road holds on that road.
- You will never see a car exceed whatever the speed limit happens to be.

Example – FD Formal Definition

- Suppose you are watching cars drive by on a road where you don't know what the speed limit is.
- At some point in time, there might be 3 cars on the road, one going 45, another going 30, and another going 42.
  - These do not *satisfy* a speed limit of 25, 10, etc.
  - We can conclude, therefore, that the speed limit is not 25.
  - They do, however, satisfy a speed limit of 55, 60, 45, etc.
  - We cannot conclude however, that, for example, 55 is the speed limit, just by looking at the cars.
  - Speed limit could be 45, 46, 47, 90, etc.

- If a particular speed limit holds on a road, then the speed of all cars on that road satisfy the speed limit.
  - Cars are like rows in a table
  - FDs that hold are like speed limits

Example – FD Formal Definition

- Consider the following relation:

A	B
1	4
1	5
3	7

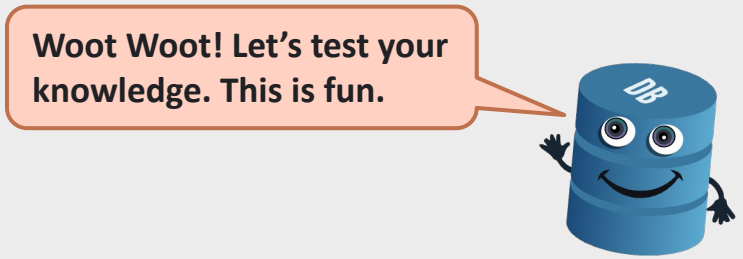
- For this set of tuples:
  - $A \rightarrow B$ is **NOT** satisfied
  - $A \rightarrow B$ therefore does **NOT** hold
  - $B \rightarrow A$ **IS** currently satisfied
  - but does $B \rightarrow A$ hold?

Example – FD Formal Definition

- Consider the following relation R (A, B, C, D):

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

- Which of the following FDs are satisfied by this relation? How?
  - *a)* $A \rightarrow B$
  - *b)* $A \rightarrow CD$
  - *c)* $AB \rightarrow CD$
  - *d)* $C \rightarrow D$
  - *e)* $B \rightarrow A$
  - *f)* $BD \rightarrow AC$
  - *g)* $AD \rightarrow BC$
  - *h)* $D \rightarrow B$
  - *i)* $D \rightarrow C$
  - *j)* $C \rightarrow A$



Example – FD Formal Definition

- By simply looking at the tuples in a relation, one can determine if an FD is currently satisfied or not.
- Similarly, by looking at the tuples you can determine that an FD doesn't hold, but you can never be certain that an FD does hold (for that you need to look at the set of FDs).
- Further, by simply looking at the cars on a road, one can determine if a speed limit is currently satisfied or not.
- Also, by looking at the cars you can determine that a speed limit doesn't hold, but you can never be certain that a speed limit does hold (for that you need to look at the speed limit sign).

FDs – Holding vs. Satisfying

- One more time - a specific relation may satisfy an FD even if the FD does not hold on all legal instances of that relation.
- Example #1: A specific instance of *Loan-info-schema* may satisfy:

  loan-number → *customer-name.*
- Example #2: A specific instance of *Grade-schema* may satisfy:

  course-id → *grade*
- Example #3: Suppose an instance of *Loan-info-schema* (or *Grade-schema*) is empty. What FDs does it satisfy?

Loan-info Schema

Loan-info-schema = (customer-name, loan-number, branch-name, amount)

Grade Schema

Grade-schema = (student-id, name, course-id, grade)

Defining Keys in Terms of FDs

- The notions of a superkey and a candidate key can be defined in terms of functional dependencies.

- 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 is a superkey for R, and
  - There is no set $\alpha \subset K$ such that α is a superkey.

- Note how declaring K as the primary key of the table effectively enforces the functional dependency $K \rightarrow R$

Examples of Trivial Functional Dependencies

- A functional dependency $\alpha \rightarrow \beta$ is said to be ***trivial*** if $\beta \subseteq \alpha$

- Examples:

  customer-name, loan-number $\rightarrow$ *customer-name*

  customer-name $\rightarrow$ *customer-name*

- Trivial functional dependencies are always satisfied (by every instance of a relation).

Armstrong's Axioms

- Given a set *F* of FDs, there are other FDs that are logically implied by *F*.
- For example, if $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.
- Example:

  ID# → Date-of-Birth

  Date-of-Birth → Zodiac-Sign

  ∴ ID# → Zodiac-Sign
- But there are other rules…

Armstrong's Axioms

- Armstrong's Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**

- Armstrong's axioms are *sound and complete*:
  - **Sound** – generate only functional dependencies that actually hold (only correct FDs).
  - **Complete** – generate all functional dependencies that hold (Generate all F^+).

Closure of a Set of FDs

- The set of all FDs logically implied by F is called the *closure* of F.
- The closure of F is denoted by F^+.
- Given a set F, we can find all FDs in F^+ by applying Armstrong's Axioms

Closure Example

- Consider the following:

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

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

Armstrong Axioms

if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$	**(reflexivity)**
if $\alpha \rightarrow \beta$, then $\gamma \; \alpha \rightarrow \; \gamma \; \beta$	**(augmentation)**
if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \; \gamma$	**(transitivity)**

- Some members of F^+
  - $A \rightarrow H$

    Transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$

    Augmentation of $A \rightarrow C$ with G, to get $AG \rightarrow CG$,
    then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$

    Augmentation of $CG \rightarrow I$ to get $CG \rightarrow CGI$,
    augmentation of $CG \rightarrow H$ to get $CGI \rightarrow HI$,
    and then transitivity CG →HI

Closure Example

- Note that a formal derivation (proof) can be given for each FD in F^+.

- Example: Show that $CG \rightarrow HI$ *is in* F^+:

1.	$CG \rightarrow I$	*Given*
2.	$CG \rightarrow CGI$	*Augmentation of (1) with CG*
3.	$CG \rightarrow H$	*Given*
4.	$CGI \rightarrow HI$	*Augmentation of (3) with I*
5.	$CG \rightarrow HI$	*Transitivity with (2) and (4)*

Armstrong Axioms

if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$	**(reflexivity)**
if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$	**(augmentation)**
if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$	**(transitivity)**

- Exercises:
  - Suppose $A \rightarrow B$ *and* $A \rightarrow C$. *Show* $A \rightarrow BC$.
  - *Suppose* $A \rightarrow BC$ *then* $A \rightarrow B$ *and* $A \rightarrow C$.

- By the way, what is the difference between $CG \rightarrow I$, $GC \rightarrow I$ *and* $CGC \rightarrow I$?

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

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

Class Exercise

- Consider the following:

$R = (P, Q, R, S, T, U)$

$F = \{ P \rightarrow Q$
$P \rightarrow R$
$QR \rightarrow S$
$Q \rightarrow T$
$QR \rightarrow U$
$PR \rightarrow U\}$

- Calculate some members of F^+

Armstrong Axioms

if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$	**(reflexivity)**
if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow\ \gamma\ \beta$	**(augmentation)**
if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow\ \gamma$	**(transitivity)**

Procedure for Computing F+

- To compute the closure of a set *F* of FDs (modified from the book):

```
F+ = F;
add all trivial functional dependencies to F+;
repeat
      for each functional dependency f in F+
         apply augmentation rules on f
         add the resulting functional dependencies to F+
      for each pair of functional dependencies f1 and f2 in F+
         if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F+
until F+ does not change any further;
```

- We will see an alternative procedure for this task later.

Worst case time is exponential!
Consider F = {
A→*B1*,
A→*B2*,
…,
A→*Bn*}

Additional FD Rules

- The following additional rules will occasionally be helpful:
  - $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ implies $\alpha \rightarrow \beta\gamma$ **(union)**
  - $\alpha \rightarrow \beta\gamma$ implies $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ **(decomposition)**
  - $\alpha \rightarrow \beta$ and $\gamma\ \beta \rightarrow \delta$ implies $\alpha\ \gamma \rightarrow \delta$ **(pseudotransitivity)**

- Notes:
  - The above rules are **NOT** Armstrong's axioms.
  - The above rules can be proven using Armstrong's axioms.

Proving the Decomposition Rule

- Example - Proving the decomposition rule.

- Suppose $\alpha \rightarrow \beta\,\gamma$. Show that $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.

  1. $\alpha \rightarrow \beta\,\gamma$ *Given*
  2. $\beta\gamma \rightarrow \beta$ *Reflexivity*
  3. $\alpha \rightarrow \beta$ *Transitivity with (1) and (2)*
  4. $\beta\gamma \rightarrow \gamma$ *Reflexivity*
  5. $\alpha \rightarrow \gamma$ *Transitivity with (1) and (4)*

Armstrong Axioms

if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$	**(reflexivity)**
if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$	**(augmentation)**
if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$	**(transitivity)**

- Exercise: prove the union rule and the pseudo-transitivity rule.

Closure of Attribute Sets

- Let α be a set of attributes, and let F be a set of functional dependencies.
- The *closure* of α *under* F (denoted by α^+) is the set of attributes that are functionally determined by α under F.
- Closure of a set of attributes α^+ is NOT the same as the closure of a set of FDs F^+.

Closure of Attribute Sets

- Algorithm to compute α^+

```
result := α;
while (changes to result) do
    for each β → γ in F do
      begin
        if β ⊆ result then result := result ∪ γ;
      end;
```

Example of Attribute Set Closure

- Consider the following:

$R = (A, B, C, D, E, F)$

$F = \{$

$A \rightarrow B,$

$CG \rightarrow H,$

$A \rightarrow C,$

$CG \rightarrow I,$

$B \rightarrow H$

$\}$

- Compute $\{A,G\}^+$

$result = AG$	
$result = ABG$	$A \rightarrow B$
$result = ABCG$	$A \rightarrow C$
$result = ABCGH$	$CG \rightarrow H$
$result = ABCGHI$	$CG \rightarrow I$

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Test Your Knowledge 1

- Consider the following:

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

$F = \{$

$AB \rightarrow C,$

$BC \rightarrow AD,$

$D \rightarrow E,$

$CF \rightarrow B$

$\}$

- Compute $\{A,B\}^+$

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Test Your Knowledge 2

- Consider the following:

$R = (A, B, C, D, E)$

$F = \{$

$A \rightarrow B,$

$B \rightarrow C,$

$C \rightarrow D,$

$D \rightarrow E$

$\}$

- Compute $\{A\}^+$
- Is $\{A,B\}^+$ a candidate key?

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Example of Attribute Set Closure

- Consider the following:

  R = (course, year, teacher, date_of_birth, age)

  F = {

  course, year → teacher

  teacher → date_of_birth

  year, date_of_birth → age

  }

Compute {year, teacher}+

course	year	teacher	date_of_birth	age
Databases	2019	Jim Jones	1974-10-12	45
Mathematics	2019	Robert Taylor	1985-05-17	34
Databases	2020	Jennifer Ohr	1990-06-09	30

Note that Mr. DB does not recommend this design in the real world.

Example of Attribute Set Closure

- Consider the following:

R = (stud_id, stud_name,
stud_phone, stud_state,
stud_country,stud_age)
F = {
stud_id → stud_name,
stud_id → stud_phone,
stud_id → stud_state,
stud_id → stud_country,
stud_id → stud_age,
stud_state → stud_country
}

stud_id	stud_name	stud_phone	stud_state	stud_country	stud_age
901411121	David	321-755-8989	Georgia	US	20
901411122	Phillip	321-098-8973	Florida	US	19
901411123	Taylor	321-900-1212	South Carolina	US	18
901411124	James	-	Florida	US	21

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Compute $\{stud\_id\}^+$

Is {stud_id, stud_name} a candidate key?

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

1. Testing if a functional dependency $\alpha \rightarrow \beta$ holds, i.e., is it in F^+ :
   - Check if $\beta \subseteq \alpha^+$
   - Is AG $\rightarrow$ I in F^+ for the previous example?

2. Testing if a set of attributes α is a superkey:
   - Check if $\alpha^+ = R$

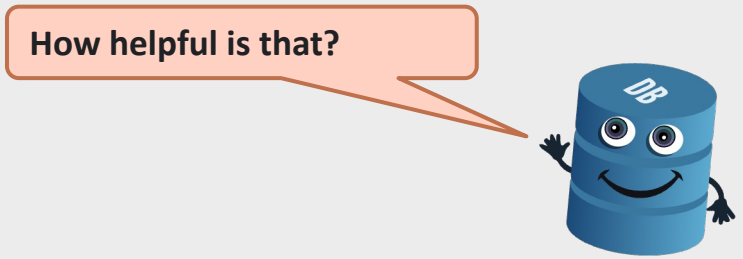
3. Testing if a set of attributes α is a candidate key:
   - Check if α^+ is a superkey (using the above)
   - Check if it has a subset $\alpha' \subset \alpha$ that is a superkey (using the above)

$R = (A, B, C, G, H, I$
$F = \{$
$A \rightarrow B,$
$CG \rightarrow H,$
$A \rightarrow C,$
$CG \rightarrow I,$
$B \rightarrow H$
$\}$

Uses of Attribute Closure

4. Computing closure of a set *F* of functional dependencies:
   - for each $\gamma \subseteq R$, we find the closure γ^+, and then
   - for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$

For example, given $\gamma^+ = \{A,B,C\}$;
n = 2^3 = 8 (powerset);
S = {} # Do we need this?
S = {A}
S = {B}
S = {C}
S = {A, B}
S = {A, C}
S = {B, C}
S = {A,B,C}



Example of Attribute Set Closure

- Is AG a candidate key for the preceding relational scheme?

  1. Is {A,G} a super key?
     - Does $\{A,G\} \rightarrow R$, i.e., is $R \subseteq \{A,G\}^+$
  2. Is any subset of {A,G} a super key?
     - Does $A \rightarrow R$, i.e., is $R \subseteq \{A\}^+$
     - Does $G \rightarrow R$, i.e., is $R \subseteq \{G\}^+$

- Is CG a candidate key?

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

$F = \{$

$A \rightarrow B,$

$CG \rightarrow H,$

$A \rightarrow C,$

$CG \rightarrow I,$

$B \rightarrow H$

$\}$

Equal Sets of FDs

- Let F_1 and F_2 be two sets of functional dependencies.
- F_1 and F_2 are said to be *equal* (or identical), denoted $F_1 = F_2$, if:
  - $F_1 \subseteq F_2$ and
  - $F_2 \subseteq F_1$
- The above definition is not particularly helpful; it merely states the obvious…

Equivalent Sets of FDs

- F_2 is said to *imply* F_1 if $F_1 \subseteq F_2^+$

- F_1 and F_2 are said to be *equivalent* , denoted $F_1 \approx F_2$, if F_1 implies F_2 and F_2 implies F_1,
- i.e.,
  - $F_2 \subseteq F_1^+$
  - $F_1 \subseteq F_2^+$

Equivalent Sets of FDs

- Consider the following sets of FDs:

$$F_1 = \{A \rightarrow B,\ B \rightarrow C,\ AB \rightarrow C\}$$
$$F_2 = \{A \rightarrow B,\ B \rightarrow C,\ A \rightarrow C\}$$

- Clearly, F_1 and F_2 are not equal.
- However, F_1 is implied by F_2 since $F_1 \subseteq F_2^+$
- And F_2 is implied by F_1 since $F_2 \subseteq F_1^+$
- Hence, F_1 and F_2 are equivalent, i.e., $F_1 \approx F_2$.

Equivalent Sets of FDs

- Consider the following sets of FDs:

  $F_1 = \{A \rightarrow B,\ CG \rightarrow I,\ B \rightarrow H,\ A \rightarrow H\}$

  $F_2 = \{A \rightarrow B,\ CG \rightarrow H,\ A \rightarrow C,\ CG \rightarrow I,\ B \rightarrow H\}$

- Clearly, F_1 and F_2 are not equal.

- However, F_1 is implied by F_2 since $F_1 \subseteq F_2^+$

- But F_2 is not implied by F_1 since $F_2 \not\subseteq F_1^+$

- Hence, F_1 and F_2 are not equivalent.

Redundant FDs

- A set of FDs may contain redundancies.
- Sometimes an entire FD is redundant:

$$A \rightarrow C \text{ is redundant in } \{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C\}$$

Redundant Attributes in FDs

- Other times, an attribute in an FD may be redundant:

$\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to
$\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$

$\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to
$\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$

Extraneous Attributes

- An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies.

- Let *F* be a set of FDs and suppose that $\alpha \rightarrow \beta$ is 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 functional dependencies
    $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F*.

Examples of Extraneous Attributes

- Example #1:

- $F = \{AB \rightarrow CD,\ A \rightarrow E,\ E \rightarrow C\ \}$

  Is C extraneous in $AB \rightarrow CD$?

- Example #2:

  $F = \{A \rightarrow C,\ AB \rightarrow CD\}$

  Is C extraneous in $AB \rightarrow CD$?

  How about A, B or D?

Is C extraneous in $AB \rightarrow CD$?

1st Approach:

$\{AB \rightarrow CD,\ A \rightarrow E,\ E \rightarrow C\} - \{AB \rightarrow CD\} \cup \{\alpha \rightarrow (\beta - C)\}$

$\{A \rightarrow E,\ E \rightarrow C\} \cup \{(AB \rightarrow (CD - C\} = \{A \rightarrow E,\ E \rightarrow C\ ,\ AB \rightarrow D\}$

A is *extraneous* in α if $A \in \alpha$
and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

A is *extraneous* in β if $A \in \beta$
and the set of functional dependencies
$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F

2ND Approach: Is CD in AB^+

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Canonical Cover – Formal Definition

- Intuitively, a canonical cover for F is a "minimal" set that is equivalent to F, i.e., having no redundant FDs, or FDs with redundant attributes.

- More formally, a *canonical cover* for F is a set of dependencies F_c such that:
  - $F \approx F_c$
  - No functional dependency in F_c contains an extraneous attribute.
  - Each left side of a functional dependency in F_c is unique.

Computing a Canonical Cover

- Given a set *F* of FDs, a canonical cover for *F* can be computed as follows:

```
repeat
    Replace any dependencies of the form α1 → β1 and α1 → β2 with α1 → β1 β2; // union rule
    Find a functional dependency α → β with an extraneous attribute either in α or in β;
    If an extraneous attribute is found, delete it from α → β;
until F does not change;
```

- Note that the union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied.

Example of Computing a Canonical Cover

$R = (A, B, C)$
$F = \{A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C\}$

- Combining $A \rightarrow BC$ and $A \rightarrow B$ gives $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$ gives $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$ gives $\{A \rightarrow B, B \rightarrow C\}$

Review – The Goals of Normalization

Recall:

- Given a relational scheme R and an associated set F of FDs, first determine whether or not R is sufficiently normalized.

- If R is not sufficiently normalized, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - Each relation is sufficiently normalized
  - The decomposition is a lossless-join decomposition
  - All dependencies are preserved

- All of the above requirements will be based on functional dependencies.

Decomposition

- Previously, we decomposed the *Lending-schema* into:

  Branch-schema = (branch-name, branch-city, assets)

  Loan-info-schema = (customer-name, loan-number, branch-name, amount)

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- The decomposition must have a lossless-join, i.e., for all possible relations *r* on *R:*

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- Having defined FDs, we can now define the conditions under which a decomposition has a loss-less join…

Decomposition

- **Theorem:** A decomposition of R into R_1 and R_2 has a lossless join if and only if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  - Can be decomposed in three different ways (with a common attribute).

- $R_1 = (A, B)$, $R_2 = (B, C)$
  - Has a lossless-join:

$$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$

- $R_1 = (A, B)$, $R_2 = (A, C)$
  - Has a lossless-join:

$$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

- $R_1 = (A, C)$, $R_2 = (B, C)$
  - Does not have a lossless-join. $R_1 \cap R_2 = \{C\}$

Recall the Zodiac-Sign FDs

ID# → Date-of-Birth

Date-of-Birth → Zodiac-Sign

∴ ID# → Zodiac-Sign

R = (ID, DOB, Zodiac)

R_1 *= (ID, Zodiac),* ***R_2*** *= (DOB, Zodiac)*

$R_1 \cap R_2$ = Zodiac;

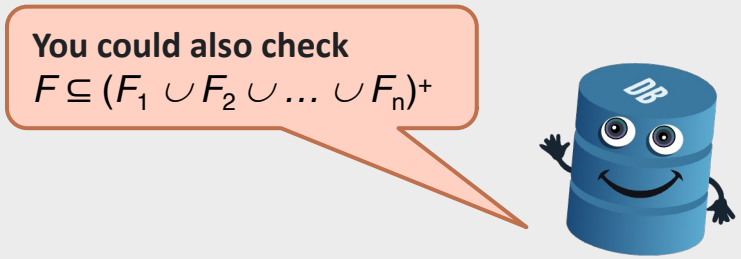
Zodiac does not functionally determine anything in the set of FDs

Preservation of Functional Dependencies

- Suppose that:
  - R is a relational scheme
  - F is an associated set of functional dependencies
  - $\{R_1, R_2, ..., R_n\}$ is a decomposition of R
  - Let F_i be the set of dependencies F^+ that include only attributes in R_i.

- The decomposition $\{R_1, R_2, ..., R_n\}$ is said to be *dependency preserving* if $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$

- Why is it important for a decomposition to preserve dependencies?
  - The goal is to replace R by $R_1, R_2, ..., R_n$
  - Enforcing $F_1, F_2, \ldots, F_n$ on $R_1, R_2, ..., R_n$ must be equivalent to enforcing F on R.



Example 1

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in three different ways.

- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition (as noted previously)
  - Dependency preserving

- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition (as noted previously)
  - Not dependency preserving; $B \rightarrow C$ is not preserved

- $R_1 = (A, C), \quad R_2 = (B, C)$
  - Does not have a lossless-join (as noted previously)
  - Not dependency preserving; $A \rightarrow B$ is not preserved

Example 2

- *R = (A, B, C, D, E)*
  F = {AB→ CD, C → D, D → E)

- *Check this decomposition* $R_1 = (A, B, C)$, $R_2 = (C, D)$, $R_3 = (D, E)$
  - Lossless-join decomposition (?)
  - Dependency preserving (?)

$AB \rightarrow CD$ *gives*	$AB \rightarrow C$ *and* $AB \rightarrow D$	*(decomposition rule)*	F_1
$AB \rightarrow C$ *can be derived from R1*			F_1
$C \rightarrow$ D *can be derived from R2*			F_2
$AB \rightarrow$ D *can be derived from both R1 and R2*		*(transitivity rule)*	
$D \rightarrow$ E *can be derived from R3*			F_3

Boyce-Codd Normal Form

A relational scheme R is in **BCNF** with respect to a set F of functional dependencies if for all functional dependencies 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 (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

Testing for BCNF

- To determine if a relational scheme is in BCNF:

  Calculate F^+

  For each non-trivial functional dependency $\alpha \rightarrow \beta$ in F^+

  1. Compute α^+ (the attribute closure of α)
  2. Verify that α^+ includes all attributes of R, i.e., that it is a superkey for R

  $\Rightarrow$ If a functional dependency $\alpha \rightarrow \beta$ in F^+ is identified that

  - $\Rightarrow$ (1) it is non-trivial ($\beta \not\subseteq \alpha$) or
  - $\Rightarrow$ (2) α is *not* a superkey ($\alpha^+ \neq R$)
  - $\Rightarrow$ then R is **not** in BCNF.

Computing α^+

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Example 1

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  Candidate Key = $\{A\}$

- R is not in BCNF *(why not?)*

Computing α^+

```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

- Decompose R into $R_1 = (A, B)$ and $R_2 = (B, C)$
  - R_1 is in BCNF
  - R_2 is in BCNF
  - The decomposition has a lossless-join (noted previously)
  - The decomposition preserves dependencies (noted previously)

Example 2

- *Is the following in BCNF?*
- *R = (A,B,C,D)*
- *F =*

  {AB → C,

  B → D,

  C → A

  }
- *R* is not in BCNF *(why not?)*

Computing α^+

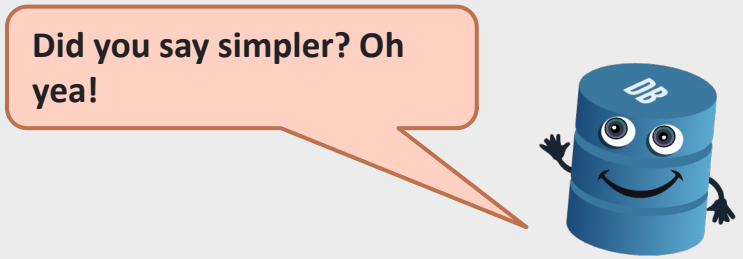
```
result := α;
while (changes to result) do
 for each β → γ in F do
  begin
    if β ⊆ result then
       result := result ∪ γ;
  end;
```

Testing for BCNF

- It turns out to be only necessary to check the dependencies in F (and not F^+).

- This leads to the following simpler definition for BCNF.

  Let R be a relational scheme and let F be a set of functional dependences. Then R is said to be in BCNF with respect to F if, for each $\alpha \rightarrow \beta$ in F, either $\alpha \rightarrow \beta$ is trivial or α is a superkey for R.



Testing for BCNF

- Note that when testing a relation R_i in a decomposition for BCNF, however, make sure you consider ALL dependencies in F_i.

- For example, consider $R = (A, B, C, D)$, with $F = \{A \rightarrow B, B \rightarrow C\}$
  - Decompose R into $R_1(A,B)$ and $R_2(A,C,D)$
  - One might think F_2 is empty, and hence R_2 satisfies BCNF.
  - In fact, $A \rightarrow C$ is in F^+, and hence in F_2, which shows R_2 is not in BCNF.

BCNF Decomposition Algorithm

- Let R be a relational scheme, let F be an associated set of functional dependencies, and suppose that R is not in BCNF.

- The following will give a decomposition of R into R_1, R_2, ..., R_n such that each R_i is in BCNF, and such that the decomposition has a lossless-join.

```
result := {R};
compute F+;
while (there is a schema Ri in result that is not in BCNF) do
   let α → β be a nontrivial functional dependency that
   holds on Ri such that α → Ri is not in F+, and α ∩ β  = ∅;
   result := (result – Ri ) ∪ (Ri – β) ∪ (α, β );
end;
```

Example of BCNF Decomposition

- Is the following Relational Scheme in BCNF?

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Canadian	Production and planning	D001	200
1001	Canadian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

F = {
emp_id → emp_nationality
emp_dept → {dept_type, dept_no_of_emp}
}

Candidate key: {emp_id, emp_dept}

Example of BCNF Decomposition

emp_nationality

emp_id	emp_nationality
1001	Canadian
1002	American

emp_dept

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

F = {
emp_id → emp_nationality
emp_dept → {dept_type, dept_no_of_emp}
}

Candidate Keys:
emp_nationality: emp_id
emp_dept : emp_dept
emp_dept_mapping: {emp_id, emp_dept}

When would the trivial rule apply?
emp_id, emp_nationality → emp_nationality
emp_id→ emp_id

Another Trivial example
{student_id, student_Name} → student_id

Keys Created by the BCNF Algorithm

- What are the primary keys for the resulting relations?
- Ideally, each R_i represents one functional dependency, where the LHS will be the primary key, i.e., α ; thus, the primary key constraint enforces the FD.
- Although this enforces the majority of FDs, it does not enforce all FDs, in general.
- In such cases the other FDs can frequently be enforced by a secondary key; in the worst case, code must be written to repeatedly check for FD violations.

Keys Created by the BCNF Algorithm

- Example:

$R = (A,B,C)$

$F = \{ A \rightarrow C,$
$B \rightarrow C,$
$A \rightarrow B,$
$B \rightarrow A\}$

Two Candidate Keys = $\{A\}$ $\{B\}$

Primary Key - A

Secondary (unique) Key - B

BCNF and Dependency Preservation

- As noted, the algorithm produces a set of BCNF relational schemes that have a lossless join, but what about preserving dependencies?

- It is not always possible to get a BCNF decomposition that is dependency preserving:

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
  Two candidate keys = JK and JL

- In terms of the banking enterprise:
  Banker-schema = (branch-name, customer-name, banker-name)
  customer-name, branch-name → banker-name
  banker-name → branch-name

- R is not in BCNF *(why?)*

BCNF and Dependency Preservation

- However, any decomposition of R will fail to preserve $JK \rightarrow L$.

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$

  Two candidate keys = JK and JL

- Decompositions:

JK	KL	J	KL
JK	JL	K	JL
JL	KL	L	JK

- In every case $JK \rightarrow L$ is lost.

Third Normal Form Motivation

- It follows that there is *no* algorithm for decomposing a relational scheme that guarantees both, i.e., BCNF and preservation of dependencies.
- Solution - Define a weaker normal form, called *Third Normal Form*.
  - Allows some redundancy (with resultant problems; as we shall see)
- *Given any relational scheme, there is always a lossless-join, dependency-preserving decomposition into 3NF relational schemes.*
- This is why 3NF is industry standard.

Third Normal Form

- A relation schema R is in third normal form (3NF) with respect to a set F of functional dependencies if, for all functional dependencies 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 (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.
- For the last condition, each attribute may be in a different candidate key.
- The third condition is a minimal relaxation of BCNF that will ensure dependency preservation.
- If a relation is in BCNF it is in 3NF *(why?)*

Testing for 3NF

- As with BCNF, the definition can be simplified to only consider FDs in F.
- The 3NF test is a slight modification of the BCNF test.
- If $\alpha \rightarrow \beta$ is not trivial, and if α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R.*
  - Expensive - requires finding all candidate keys.
  - Testing for 3NF has been shown to be NP-hard, i.e., likely requires exponential time.
  - Ironically, decomposition into third normal form (described shortly) can be done in polynomial time.

BCNF vs. 3NF

- Note that our previous “problematic” scheme is in 3NF but not BCNF:

$R = (J, K, L)$
$F = \{JK \rightarrow L, L \rightarrow K\}$

Two candidate keys = JK and JL

BCNF vs 3NF

manager	project	branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Mars	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

3NF: One of the following must be satisfied

1. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
2. α is a superkey for R
3. Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

- Each manager works in a particular branch
- Each project has several managers, and runs on several branches; however, a project has a unique manager for each branch.

- F = {manager → branch,
  project, branch → manager
  }

3NF Decomposition Algorithm

- 3NF Decomposition Algorithm:

```
Let F_c be a canonical cover for F;
i := 0;
for (each functional dependency α → β in F_c) loop
  if (none of the schemas R_j, 1 ≤ j ≤ i contains α and β) then
     i := i + 1;
     R_i := (α,β);
  end if;
end loop;
if (none of the schemas R_j, 1 ≤ j ≤ i contains a candidate key for R) then
  i := i + 1;
  R_i := any candidate key for R;
end if;
return (R_1, R_2, ..., R_i);
```

- Each resulting R_i is in 3NF, the decomposition has a lossless-join, and all dependencies are preserved.

- Each resulting R_i represents one or more functional dependencies, *one* of which will be enforced by a primary key.

Example 1

- Relation schema *R*:

  Banker-schema = (branch-name, customer-name, banker-name, office-number)

- Functional dependencies *F*:

  banker-name $\rightarrow$ *branch-name, office-number*
  customer-name, branch-name $\rightarrow$ *banker-name*

- Candidate keys:

  {customer-name, branch-name}

  {customer-name, banker-name}

3NF: One of the following must be satisfied

1. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
2. α is a superkey for *R*
3. Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*.

- *R* is not in 3NF *(why?)*

- The algorithm creates the following schemas (*F* is already a canonical cover):

  Banker-office-schema = (banker-name, branch-name, office-number)
  Banker-schema = (customer-name, branch-name, banker-name)

Example 2

- Given the following schema, is it in 3NF:
  - vendor = (id, name, account_no, bank_code_no, bank_name)

Candidate key = {ID}

F = {id $\rightarrow$ {name, account_no, bank_code_no},
bank_code_no $\rightarrow$ bank_name
}

- Break vendor into vendor and bank:

vendor = (id, name, account_no, bank_code_no}

bank = (bank_code_no, bank_name)

3NF: One of the following must be satisfied

1. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
2. α is a superkey for R
3. Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

Second Normal Form (2NF) - Historical

- A relation schema R is in second normal form if every non-prime attribute A in R is fully functionally dependent on the primary key.
- More formally, A functional dependency $\alpha \rightarrow \beta$ is called a partial dependency if there is a proper subset γ of α such that $\gamma \rightarrow \beta$; we say that β is partially dependent on α. A relation schema R is in second normal form (2NF) if each attribute A in R meets one of the following criteria:
  - It appears in a candidate key
  - It is not partially dependent on a candidate key

- **Prime attribute**: an attribute that is a member (part) of a candidate key
- **Partial dependency**: a non-key attributes only depends on a subset of the primary key

Second Normal Form (2NF) - Historical

- R = (A, B, C, D)
- F = {AB → C
  BC → D
  }

Candidate key = {AB}

- In the above relation, AB is the only candidate key and there is no partial dependency, (i.e., no proper subset of AB determines any non-prime attributes)

Example

- Given the following relational schema R, note that R is not in 2NF

teacher_id	subject	teacher_age
111	Math	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

- **Candidate Keys**: {teacher_id, subject}
- **F** ={ teacher_id, subject → teacher_age,
  teacher_id → teacher_age}

Example

- Decompose *R into two tables*
- **Candidate Keys**: {teacher_id, subject} and {teacher_id}

teacher_details

teacher_id	teacher_age
111	38
222	38
333	40

F ={teacher_id → teacher_age}

teacher_subjects

teacher_id	subject
111	Math
111	Physics
222	Biology
333	Physics
333	Chemistry

Summary: Comparison of BCNF and 3NF

- In summary…
- It is always possible to decompose a relational scheme into a set of relational schemes such that:
  - All resulting relational schemes are in 3NF
  - The decomposition has a lossless join
  - All dependencies are preserved

- It is always possible to decompose a relational scheme into a set of relational schemes such that:
  - All resulting relational schemes are in BCNF
  - The decomposition has a lossless join

  => The decomposition, however, is not guaranteed to preserve dependencies.

3NF (Cont.)

Note #1:

- So how does 3NF help us with our “problem” schema?

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$

  Two candidate keys: JK and JL

- Although R is not in BCNF, it is in 3NF:

  $JK \rightarrow L$ JK is a superkey
  $L \rightarrow K$ K is contained in a candidate key

- In other words, if 3NF is our desired level of normalization, then the new algorithm leaves it as is.

Summary: Comparison of BCNF and 3NF, Cont.

- But there is a “cost” to accepting this schema as is…

- Redundancy in 3NF:

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

Design Goals

Note #2:

- It is relatively easy to prove that if a relational scheme is in 3NF but not in BCNF, such a relational scheme must have multiple distinct overlapping candidate keys (left as an exercise).

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
  Two candidate keys = JK and JL

- Thus, if a relational scheme does not have multiple distinct overlapping candidate keys, and if it is in 3NF, then it is also in BCNF.

- Another reason why 3NF is industry standard.

Design Goals

Note #3:

- SQL does not provide a direct way of specifying functional dependencies other than as primary or secondary keys.
- So how are the FDs in the following enforced (in particular, the second)?

  $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$

- FDs can be specified using assertions, but they are expensive to test.
- FDs can also be checked in program code, but that has drawbacks.
- In general, using SQL there is no efficient way to test a functional dependency whose left-hand side is not a key.

End of Chapter