THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

5a. Functional Dependencies

CSCI 2541W
Database Systems & Team Projects

Gabe

Good and Bad Schemas

Functional Dependencies

Normal Forms based on Functional Dependencies

Normal Form Examples: 1NF

1NF: Attributes should be atomic and tables should have no repeating groups

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

BAD: One cell should not have multiple values in it!

Customer ID	First Name	Surname	TNumber1	TNumber2
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	

BAD: Should not have repeating columns!

Normal Form Examples: 2NF

2NF: No value in a table should be dependent on only **part** of a key that uniquely identifies a row

1NF

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	Zhang	555-808-9633

BAD: First/Last name depend on only Customer ID

VS

2NF

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

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

Meets 2NF

Normal Form Examples: 3NF

3NF: No value should be able to be dependent on another non-key field

Customer ID	First Name	Surname	Birthday	Age	Fav Color
123	Pooja	Singh	1/4/1984	37	Blue
456	San	Zhang	3/15/2001	19	Blue
789	John	Zhang	11/12/2006	14	Buff

BAD: Age is based on Birthday (non-key)

Tournament	Year	Winner	Winner's Birthplace
Indiana Invitational	1998	Al Fredrickson	Ohio
Cleveland Open	1999	Bob Albertson	New York
Des Moines Masters	1999	Al Fredrickson	Ohio
Indiana Invitational	1999	Chip Masterson	Kentucky

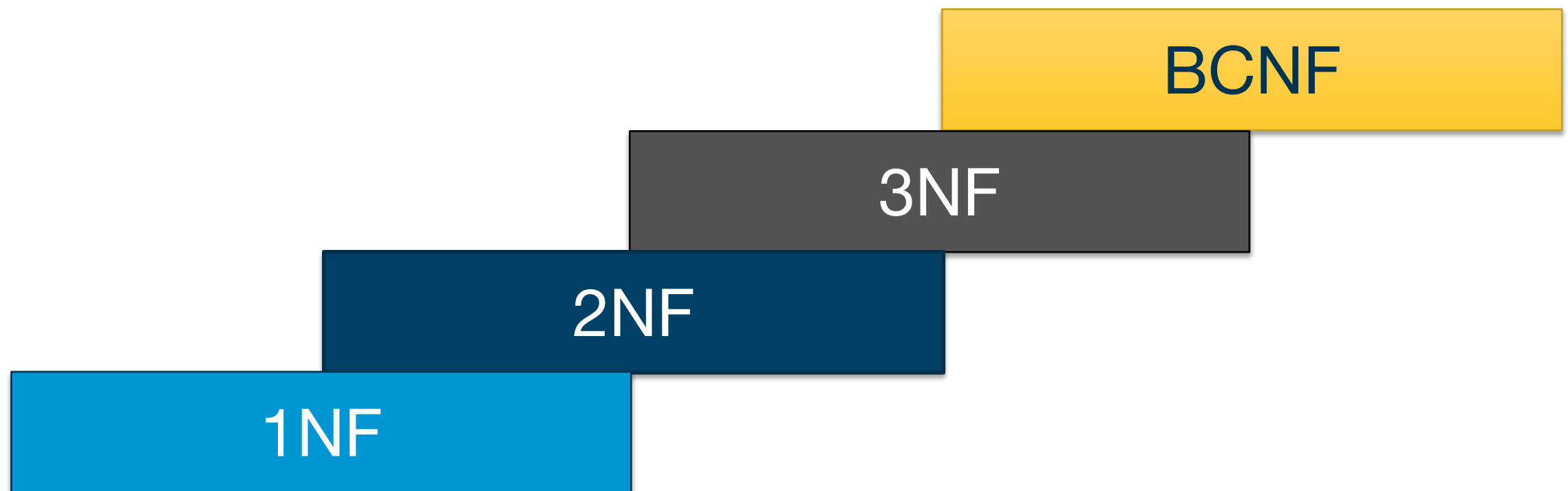**BAD:** Birthplace based on Winner (non-key)

Summary

1NF: ensures atomicity of cells and prevents repetition of identical column types

2NF: prevents data across rows

3NF: prevents repetition of data within+across rows

Dependencies

How can we **formally represent dependencies** between Attributes in a Relation?

BCNF

3NF

2NF

1NF

Functional Dependencies

Use **functional dependencies!** (abbreviated **FD**)

We say a set of attributes **X** functionally determines an attribute **Y** if *given the values of **X** we always know the only possible value of **Y***.

- Notation: **X** → **Y**
- X **functionally determines** Y
- Y is **functionally dependent** on X

Example:

- GWID → Name
- {GWID, CourseID, Semester, Year} → Grade

Sets of Functional Dependencies

Some more functional dependencies

- {GWID} → {NAME, ADDRESS, MAJOR}
- {MAJOR} → {DEPT_NAME, DEPT_CHAIR}

From above dependencies, we can infer

- {GWID} → {DEPT_NAME, DEPT_CHAIR}

We can do math on functional dependencies!

A functional dependency "holds" if it must be true for all legal relations

Functional Dependency Ops

Armstrong's Axioms: where A, B, C are sets of attributes

- **Reflexive rule**: if B ⊆ A, then A → B (if B is subset of A)

- **Augmentation rule**: if A → B, then C ∪ A → C ∪ B

- **Transitivity rule**: if A → B, and B → C, then A → C

These rules are

- Sound and complete — generate all functional dependencies that hold.

{GWID} → {Name, Address, Major}

{Major} → {Dept_Name, Dept_Chair}

{GWID, CourseID, Semester, Year} → Grade

Functional Dependency Ops

Armstrong's Axioms: where A, B, C are sets of attributes

- **Reflexive rule**: if B ⊆ A, then A → B (if B is subset of A)

- **Augmentation rule**: if A → B, then C ∪ A → C ∪ B

- **Transitivity rule**: if A → B, and B → C, then A → C

Given:

{GWID} → {Name, Address, Major}

{Major} → {Dept_Name, Dept_Chair}

{GWID, CourseID, Semester, Year} → Grade

R:{GWID, CourseID, Semester, Year} → {GWID, Year}

A:{Major, Name, Address} →

　　　　　　{Dept_Name, Dept_Chair, Name, Address}

T:{GWID} → {Name, Address, Major} →

　　　　　　{Dept_Name, Dept_Chair, Name, Address}

Functional Dependency Ops

Armstrong's Axioms: where A, B, C are sets of attributes

— **Reflexive rule**: if B ⊆ A, then A → B (if B is subset of A)

— **Augmentation rule**: if A → B, then C ∪ A → C ∪ B

— **Transitivity rule**: if A → B, and B → C, then A → C

These rules are

— Sound and complete — generate all functional dependencies that hold.

Bonus rules to make life easier:

— **Union rule**: If A → B holds and A → C holds, then A → B ∪ C holds.

— **Decomposition rule**: If A → B ∪ C holds, then A → B holds and A → C holds.

— **Pseudotransitivity rule**:If A → B holds and C ∪ B → D holds, then A ∪ C → D holds.

Functional Dependency Ops

Armstrong's Axioms: where A, B, C are sets of attributes

- **Reflexive rule**: if B ⊆
- **Augmentation rule**:
- **Transitivity rule**: if A

These rules are
- Sound and complete —
  hold.

> From now on:
> will *shorthand* B ∪ C as BC

Bonus rules to make life easier:
- **Union rule**: If A → B holds and A → C holds, then A → B ∪ C holds.
- **Decomposition rule**: If A → B ∪ C holds, then A → B holds and A → C holds.
- **Pseudotransitivity rule**: If A → B holds and C ∪ B → D holds, then A ∪ C → D holds.

Definition: Closure of a Set of FD's

Defn. Let F be a set of FD's.

Its **closure**, **F+**, is the set of all FD's:

$$\{X \to Y \mid X \to Y \text{ is derivable from F by Armstrong's Axioms}\}$$

Two sets of dependencies F and G are equivalent if F+=G+

- i.e., their closures are equal
- i.e., the same sets of FDs can be inferred from each

Example Closure

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

F = { A → B

A → C

CG → H

CG → I

B → H}

Reflexive rule: if β ⊆ α, then α → β

Augmentation rule: if α → β, then γ α → γ β

Transitivity rule: if α → β, and β → γ, then α → γ

Example Closure

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

  F = { A → B

   A → C

   CG → H

   CG → I

   B → H}

Reflexive rule: if β ⊆ α, then α → β

Augmentation rule: if α → β, then γ α → γ β

Transitivity rule: if α → β, and β → γ, then α → γ

A few members of F+ include:

— A → H

— AG → I

— CG → HI

Example Closure

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

F = { A → B

A → C

CG → H

CG → I

B → H}

Reflexive rule: if β ⊆ α, then α → β

Augmentation rule: if α → β, then γ α → γ β

Transitivity rule: if α → β, and β → γ, then α → γ

A few members of F+ include:

－ A → H

by transitivity from A → B and B → H

－ AG → I

－ CG → HI

Example Closure

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

   F = { A → B

    A → C

    CG → H

    CG → I

    B → H}

Reflexive rule: if β ⊆ α, then α → β

Augmentation rule: if α → β, then γ α → γ β

Transitivity rule: if α → β, and β → γ, then α → γ

A few members of F+ include:

– A → H
by transitivity from A → B and B → H

– AG → I
by augmenting A → C with G, to get AG → CG
and then transitivity with CG → I

– CG → HI

Example Closure

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

   F = { A → B

    A → C

    CG → H

    CG → I

    B → H}

Reflexive rule: if β ⊆ α, then α → β

Augmentation rule: if α → β, then γ α → γ β

Transitivity rule: if α → β, and β → γ, then α → γ

A few members of F+ include:

- A → H
  by transitivity from A → B and B → H
- AG → I
  by augmenting A → C with G, to get AG → CG
  and then transitivity with CG → I
- CG → HI
  by augmenting CG → I to infer CG → CGI (note CGCG → CGI is CG → CGI),
  and augmenting of CG → H to infer CGI → HI,
  and then transitivity (CG → CGI → HI).
  This derives the *Union* rule!

Functional Dependencies and Keys

A Candidate Key is a minimal set of attributes which are sufficient to uniquely identify each tuple in a relation

- All other attributes must be functionally dependent on the set of attributes that make up the Candidate Key.

Thus a candidate key must be a minimal set of attributes which can appear on the left hand side of functional dependencies, but will produce a closure that includes all other attributes on the right hand side

$$\{A, B\} \rightarrow \{C, D, E\}$$

Functional Dependencies and Keys

A candidate key must be a minimal set of attributes which can appear on the left hand side of functional dependencies, but will produce a closure that includes all other attributes on the right hand side

What is a candidate key for R?

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

$F = \{ A \rightarrow B$

$A \rightarrow C$

$CG \rightarrow H$

$CG \rightarrow I$

$B \rightarrow H\}$

Functional Dependencies and Keys

A candidate key must be a minimal set of attributes which can appear on the left hand side of functional dependencies, but will produce a closure that includes all other attributes on the right hand side

What is a candidate key for R?

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

F = { A → B

A → C

CG → H

CG → I

B → H}

Candidate Key: (A, G)
Non-Prime Attribs:
(B,C,G,H,I)

Functional De...

A candidate key must b[e]
appear on the left hand
produce a closure that
hand side

What is a candidate key

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

F = { A → B

A → C

CG → H

CG → I

B → H}

Pseudotransitivity rule:

If A → B holds and CB → D holds,
then AC → D holds.

How?
A→C and CG→H
Augmentation: AG→CG
Transitivity: AG →CG→H

Candidate Key: (A, G)
Non-Prime Attribs:
(B,C,G,H,I)

Good and Bad Schemas

Functional Dependencies

Normal Forms based on Functional Dependencies

Redefining 2NF

Using Functional Dependencies and Closures lets us more precisely define our Normal Forms

Second Normal Form: For every $X \rightarrow A$ that holds over relationship schema **R**, where A is a non-prime attribute

(i.e., A is not an attribute in any candidate key)

1. either $A \in X$ (it is trivial), or
2. **X** is a superkey for **R**, or
3. **X** is transitively dependent on a super key R

Easier to think of the opposite: There cannot be $X \rightarrow A$ where X is a partial candidate key for R

－ Says nothing about non-prime to non-prime dependencies!

2NF Violations

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	Zhang	555-808-9633

Tournament	Year	Winner	Winner's Birthplace
Indiana Invitational	1998	Al Fredrickson	Ohio
Cleveland Open	1999	Bob Albertson	New York
Des Moines Masters	1999	Al Fredrickson	Ohio
Indiana Invitational	1999	Chip Masterson	Kentucky

2NF Violations

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	Zhang	555-808-9633

ID -> {First Name, LastName} Violates 2NF since **ID** is a partial Candidate Key

Tournament	Year	Winner	Winner's Birthplace
Indiana Invitational	1998	Al Fredrickson	Ohio
Cleveland Open	1999	Bob Albertson	New York
Des Moines Masters	1999	Al Fredrickson	Ohio
Indiana Invitational	1999	Chip Masterson	Kentucky

No 2NF violation

Redefining 3NF

No value should be able to be dependent on another non-key field

Third Normal Form (3NF): For every $X \rightarrow A$ that holds over relationship schema **R**,

1. either $A \in X$ (it is trivial), or
2. **X** is a superkey for **R**, or
3. **A** is a member of some key for **R**

Easier to think of: **X** must be a full candidate key, unless A itself is a part of a candidate key

"Every non-key attribute must provide a fact about the Key, the whole Key, and nothing but the Key… so help me Codd"

3NF Violations

Customer ID	First Name	Surname	Birthday	Age	Fav Color
123	Pooja	Singh	1/4/1984	37	Blue
456	San	Zhang	3/15/2001	19	Blue
789	John	Zhang	11/12/2006	14	Buff

Tournament	Year	Winner	Winner's Birthplace
Indiana Invitational	1998	Al Fredrickson	Ohio
Cleveland Open	1999	Bob Albertson	New York
Des Moines Masters	1999	Al Fredrickson	Ohio
Indiana Invitational	1999	Chip Masterson	Kentucky

3NF Violations

C ->
F,S,B,A,F
B->A

Customer ID	First Name	Surname	Birthday	Age	Fav Color
123	Pooja	Singh	1/4/1984	37	Blue
456	San	Zhang	3/15/2001	19	Blue
789	John	Zhang	11/12/2006	14	Buff

T,Y -> W, WB
W -> WB

Tournament	Year	Winner	Winner's Birthplace
Indiana Invitational	1998	Al Fredrickson	Ohio
Cleveland Open	1999	Bob Albertson	New York
Des Moines Masters	1999	Al Fredrickson	Ohio
Indiana Invitational	1999	Chip Masterson	Kentucky

Birthday->Age holds, but **Birthday** is not a superkey

Winner -> Birthplace holds, but **Winner** is not a superkey

Normal Forms 1-3

1NF: Attributes should be atomic and tables should have no repeating groups
- — *Prevents messiness within a cell and repetition of rows*

2NF: There cannot be $X \rightarrow A$ where X is a partial candidate key for R
- — Doesn't forbid non-prime to non-prime dependencies
- — *Prevents repetition of cells across rows*

3NF: There cannot be $X \rightarrow A$ where X is not a full candidate key for R (unless A is a Key)
- — Only allows dependencies on Keys
- — *Prevents repetition of data within a row*

Good and Bad Schemas

Functional Dependencies

Even more normal forms!

Normal Forms 1-3

1NF: Attributes should be atomic and tables should have no repeating groups

— *Prevents messiness within a cell and repetition of rows*

2NF: There cannot be **X → A** where X is a partial candidate key for R

— Doesn't forbid non-prime to non-prime dependencies
— *Prevents repetition of cells across rows*

3NF: There cannot be **X → A** where X is not a **full** candidate key for R (unless A is a Key)

— Only allows dependencies on Keys
— *Prevents repetition of data within a row*

Normal Form

Normal form reference:

— 2NF: Cannot have partial Key on left hand side (LHS)

— 3NF: Meet 2NF and LHS must be full Candidate Key or RHS must be a key

ID	First name	Cid	Subj	Num	Grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

Functional Dependencies

$ID \rightarrow FirstName$
$ID, Cid \rightarrow Num, Grade$
$Num \rightarrow Subj$

What normal form is this?

Normal Form

Normal form reference:

— 2NF: Cannot have partial Key on left hand side (LHS)

— 3NF: Meet 2NF and LHS must be full Candidate Key or RHS must be a key

ID	First name	Cid	Subj	Num	Grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

Functional Dependencies

$ID \rightarrow FirstName$

$ID, Cid \rightarrow Num, Grade$

$Num \rightarrow Subj$

partial key ID violates 2NF!

non-prime LHS would also violate 3NF!

Only meets 1NF

How to Judge Decomposition?

ID	First name	Cid	Subj	Num	Grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

R1   R2

ID → FirstName
ID, Cid → Num, Grade
Num → Subj

Lossless Decomposition test:

- **R1**, **R2** is a lossless join decomposition of **R** with respect to **F** **iff** at least one of the following dependencies is in **F+**
- **(R1 ∩ R2) → R1 – R2**
- **(R1 ∩ R2) → R2 – R1**

Lossless Decomposition

R1   R2

ID	First name	Cid	Subj	Num	Grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

ID → FirstName
ID, Cid → Num, Grade
Num → Subj

R1 ∩ R2 = CID

R1 - R2 = ID, First

R2 - R1 = Subj, Num, Grade

Lossless?

Lossless Decomposition test:

– **R1**, **R2** is a lossless join decomposition of **R** with respect to **F iff** at least one of the following dependencies is in **F+**

– **(R1 ∩ R2) → R1 – R2**

– **(R1 ∩ R2) → R2 – R1**

Lossless Decomposition

ID	First name	Cid	Subj	Num	Grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

R1 (ID, First name, Cid)

R2 (Cid, Subj, Num, Grade)

$ID \rightarrow FirstName$
$ID, Cid \rightarrow Num, Grade$
$Num \rightarrow Subj$

Lossless Decomposition test:

- **R1**, **R2** is a lossless join decom
  F iff at least one of the followi

- **(R1 ∩ R2) → R1 – R2**

- **(R1 ∩ R2) → R2 – R1**

R1 ∩ R2 = CID

R1 - R2 = ID, First

R2 - R1 = Subj, Num, Grade

Not true: CID → First

Not true: CID → Grade

Not lossless!

Dependency Preservation

We also must maintain dependences

After decomposition from **R** to **R1 … Rn,** the closure of FDs of all **R1…Rn** must be equivalent to that of **R**

R1 = <u>ID</u>, FirstName, CID

R2 = <u>CID</u>, Sub, Num, Grade

or

R3 = <u>ID</u>, FirstName

R4 = <u>ID, CID</u>, Sub, Num, Grade

ID → FirstName

ID, Cid → Num, Grade

Num → Subj

Dependency Preservation

We also must maintain dependences

After decomposition from **R** to **R1 … Rn,** the closure of FDs of all **R1…Rn** must be equivalent to that of **R**

R1 = <u>ID</u>, FirstName, CID

R2 = <u>CID</u>, Sub, Num, Grade

ID → FirstName

ID, Cid → Num, Grade

Num → Subj

or

R1,R2 will lose the FD:
ID,CID -> Num, Grade

R3 = <u>ID</u>, FirstName

R4 = <u>ID, CID</u>, Sub, Num, Grade

R3,R4 will maintain all FDs (why Sub?)

3NF

It is **always possible** to decompose a relation R into a set of relations R1…Rn which is **dependency preserving** and **lossless** that is in 3NF

3NF is the baseline for acceptable DB normalization in practice!

but 3NF is not perfect…

When does 3NF fail?

Suppose we want to store addresses:

ADDR_INFO(**CITY**, **ADDRESS**, **ZIP**)

{CITY, ADDRESS} → ZIP

{ZIP} → {CITY}

Meets 3NF since LHS is a full Key **or** RHS is a Key

3NF: There cannot be **X** → **A** where X is not a full candidate key for R **(unless A is a Key)**

When does 3NF fail?

ADDR_INFO(**CITY**, **ADDRESS**, **ZIP**)

{CITY, ADDRESS} → ZIP

{ZIP} → {CITY}

City	Address	Zip
Washington	800 22nd St NW	20052
Washington	1600 Pennsylvania Avenue NW	20050

3NF: There cannot be **X → A** where X is not a full candidate key for R **(unless A is a Key)**

When does 3NF fail?

ADDR_INFO(**CITY**, **ADDRESS**, **ZIP**)

{CITY, ADDRESS} → ZIP

{ZIP} → {CITY}

City	Address	Zip
Washington	800 22nd St NW	20052
Washington	1600 Pennsylvania Avenue NW	20050
Philadelphia	101 South Street	20050

3NF does not prevent insertion/update of tuples which violate our FDs!

3NF vs BCNF

Third Normal Form (3NF): For every $X \rightarrow A$ that holds

over relationship schema **R**,

    1. either $A \in X$ (it is trivial), or

    2. **X** is a superkey for **R**, or

    3. **A** is a member of some key for **R**

Option 3 can result in update anomalies!

Boyce-Codd Normal Form (BCNF) resolves this issue:

For every $X \rightarrow A$ that holds over relationship schema **R**,

    1. either $A \in X$ (it is trivial), or

    2. **X** is a superkey for **R**

BCNF

BCNF is stricter than 3NF

- If a relation is in BCNF, it is also in 3NF;
- if it is not in 3NF, it is not in BCNF

Note:

- There are polynomial time algorithms **guaranteed to provide a lossless, dependency preserving decomposition** into 3NF
- **but** a **dependency preserving** decomposition into BCNF **may not exist**, and no polynomial time algorithm for **lossless decomposition** is known.

Normalization Summary

Functional Dependencies: Capture the dependencies between attributes

Normalization: Provides a schema that ensures functional dependencies will be kept consistent, without losing data

Normal Forms: Try to achieve BCNF, but 3NF is OK in some cases (1NF/2NF -> bad design!)

2NF vs 3NF vs BCNF

Second Normal Form (2NF): For every $X \rightarrow A$ that holds over relationship schema **R**,

1. If **A** is a non-prime attribute, then **X** cannot be a partial Candidate Key

Third Normal Form (3NF): For every $X \rightarrow A$ that holds over relationship schema **R**,

1. either **A** ∈ **X** (it is trivial), or
2. **X** is a superkey for **R**, or
3. **A** is a member of some key for **R**

Option 3 can result in update anomalies!

Boyce-Codd Normal Form (BCNF) resolves this issue:

For every $X \rightarrow A$ that holds over relationship schema **R**,

1. either **A** ∈ **X** (it is trivial), or
2. **X** is a superkey for **R**