CS 4604: Introduction to Database Management Systems

BCNF, 3NF and Normalization

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

Today's Topics

- DB design and normalization
  - pitfalls of bad design
  - decomposition
  - normal forms

Goal

- Design ‘good’ tables
  - Define what ‘good’ means
  - Fix ‘bad’ tables
- in short: “we want tables where the attributes depend on the primary key, on the whole key, and nothing but the key”

Pitfalls

- takes1 (ssn, c-id, grade, name, address)

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main

Pitfalls

- Key: {ssn, c-id}

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Pitfalls

- ‘Bad’ - why? because: ssn->address, name

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Redundant!

Pitfalls

- Redundancy
  - space
  - (inconsistencies)
  - insertion/deletion anomalies

Pitfalls

- Insertion anomaly:
  - "jones" registers, but takes no class - no place to store his address!

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
...	...	...	...	...
234	null	null	jones	Forbes

Pitfalls

- deletion anomaly: delete the last record of ‘smith’ (we lose his grade!)

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

Solution: decomposition

- split offending table in two (or more), eg.:

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
123	211	A	smith	Main

? ?

Decompositions

- A tool that allows us to eliminate redundancy
- Lossless-Join Decomposition
- Dependency-Preserving Decomposition

Decompositions - lossy

— R1(ssn, grade, name, address) R2(c-id, grade)

Ssn	Grade	Name	Address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

c-id	Grade
413	A
415	B
211	A

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address
ssn, c-id -> grade

FDs

Decompositions - lossy

—can not recover original table with a join!

Ssn	Grade	Name	Address
123	A	smith	Main
123	B	smith	Main
234	A	jones	Forbes

c-id	Grade
413	A
415	B
211	A

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address
ssn, c-id -> grade

Example: lossy

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

A	B
1	2
4	2

B	C
2	3
2	6

Example: lossy

- R = (A,B,C); decomposed into R1(A,B); R2(B,C)

A	B	C
1	2	3
4	2	6

A	B
1	2
4	2

B	C
2	3
2	6

Nat.Join

A	B	C
1	2	3
4	2	6
1	2	6
4	2	3

We get back some “bogus tuples”!
Lossless decompositions (like BCNF) don’t give bogus tuples.

Decompositions

- example of non-dependency preserving

S#	address	status
123	London	E
125	Paris	E
234	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

Decompositions

- (drill: is it lossless?)

S#	address	status
123	London	E
125	Paris	E
234	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Pitts.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

Decompositions - lossless

- Definition: Consider schema R, with FD ‘F’. R1, R2 is a lossless join decomposition of R if we always have: $R1 \bowtie R2 = R$

- An easier criterion?

Decompositions - lossless

- Theorem: lossless join decomposition if the joining attribute is a **superkey** in at least one of the new tables
- Formally: if you are decomposing R into R1 and R2 then (so R = R1 U R2)

$$R1 \cap R2 \rightarrow R1 \; or$$

$$R1 \cap R2 \rightarrow R2$$

Decompositions - lossless

- Example

R1

Ssn	c-id	Grade
123	413	A
123	415	B
234	211	A

ssn, c-id -> grade

R2

Ssn	Name	Address
123	smith	Main
234	jones	Forbes

ssn->name, address

Ssn	c-id	Grade	Name	Address
123	413	A	smith	Main
123	415	B	smith	Main
234	211	A	jones	Forbes

ssn->name, address

ssn, c-id -> grade

Decompositions – depend. pres

- informally: we don't want the original FDs to span two tables - counterexample:

S#	address	status
123	London	E
125	Paris	E
234	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address   **S# -> status**

Decompositions – depend. pres

- dependency preserving decomposition:

S#	address	status
123	London	E
125	Paris	E
234	Blacks.	A

S# -> address, status
address -> status

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	A

S# -> address **address -> status**

(but: S#->status ?)

Decompositions – depend. pres

- informally: we don’t want the original FDs to span two tables.
- So more specifically: … the FDs of the **canonical cover**.

Decompositions – depend. pres

- why is dependency preservation good?

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

(address->status: 'lost')

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	A

S# -> address **address -> status**

Decompositions – depend. pres

- A: eg., record that ‘Blacks’ has status ‘A’

S#	address
123	London
125	Paris
234	Blacks.

S#	status
123	E
125	E
234	A

S# -> address

S# -> status

(address->status: ‘lost’)

S#	address
123	London
125	Paris
234	Blacks.

address	status
London	E
Paris	E
Blacks.	A

S# -> address **address -> status**

Decompositions – conclusion

- Decompositions should always be lossless
  - joining attribute -> superkey
- Whenever possible, we want them to be dependency preserving (not always possible to do that)

Normal Forms

- Normal forms (How to detect the problem)
  - **Boyce Codd Normal Form (BCNF)**
  - First Normal Form (1NF) = all attributes are atomic
  - Second Normal Form (2NF) = old and obsolete
  - Third Normal Form (3NF) = rarely preferred over BCNF
  - Fourth Normal Form (4NF) = unnecessary/complex
- R in BCNF is in 3NF, R in 3NF is in 2NF, R in 2NF is in 1NF

Normal Forms

- We saw how to fix ‘bad’ schemas
- But what is a ‘good’ schema?
- Answer: ‘good’, if it obeys a ‘normal form’
  - i.e., a set of rules.
- Typically: Boyce-Codd Normal Form (BCNF)
  - A simple condition for removing redundancy/ anomalies from relations

Boyce-Codd Normal Form (BCNF)

- Definition: a relation R is in BCNF wrt F, if:
  - Informally: everything depends on the full key, and nothing but the key
  - Semi-formally: every determinant i.e., the left-side (LHS) is a candidate key
  - **Formally: for every FD $A_1A_2 \ldots A_n \rightarrow B$ in F**
    - **$A_1A_2 \ldots A_n \rightarrow B$ is trivial (a superset of B) or**
    - **$A_1A_2 \ldots A_n$ is a superkey for R (if $A_1A_2 \ldots A_n \rightarrow B$ is nontrivial)**
- Non-trivial means RHS is not a subset of LHS
- "Whenever a set of attributes of R is determining another attribute, it should determine all attributes of R."

Example

Name	PID	Phone Number
Nathan	nate	(540) 231 - 1234
Nathan	nate	(540) 231 - 5678
John	john	(808) 123 - 4567
John	john	(808) 123 - 1239

- What are the dependencies? SSN → Name
- Is the left side a superkey? No
- Is it in BCNF? No

Normalization

- Theorem: given a schema R and a set of FD ‘F’, we can always decompose it to schemas R1, … Rn, so that
  - R1, … Rn are in **BCNF** and
  - The decompositions are **lossless**
  - Note: some decompositions might lose dependencies
    - Dependency-preserving is not guaranteed
- It is guaranteed that we can always decompose it to 3NF relation schemas, lossless, and dependency-preserving

Decompose it into BCNF

PID	Name
nate	Nathan
john	John

SSN → Name

PID	Phone Number
nate	(540) 231 - 1234
nate	(540) 231 - 5678
john	(808) 123 - 4567
john	(808) 123 - 1239

Decomposition into BCNF: Algorithm

- For a relation *R* with a set of FDs *F*
- Given $X \subset R$, *A* be a single attribute in *R*
  1. For every FD $X \rightarrow A$ that violates BCNF
     - Decompose *R* into ***R – A*** and ***XA***
  2. Repeat recursively
- ***R – A*** denotes the set of attributes other than A in R
- **XA** denotes the union of attributes in X and A

Example Decomposition

- R = {C,S,J,D,P,Q,V} and C is a superkey
- F = {JP → C, SD → P, J → S}
- JP is a superkey (BCNF)
- SD is not a superkey nor is P ⊆ SD, so in violation of BCNF
  - Compute SD+ = {S,D,P}
  - Decompose R into
    - R_1 = SD+ = {S,D,P}
    - R_2 = SD ∪ (R − SD+) = {S,D} ∪ {C,J,Q,V} = {S,D,C,J,Q,V}
- J → S is in violation of BCNF
  - Compute J+ = {J, S}
  - Decompose R2 into
    - R3 = J+ = {J, S}
    - R4 = J ∪ (R2 – J+) = {J} ∪ {D, C, Q, V} = {J, D, C, Q, V}
- The BCNF form of the relation is SDP, JS, JDCQV

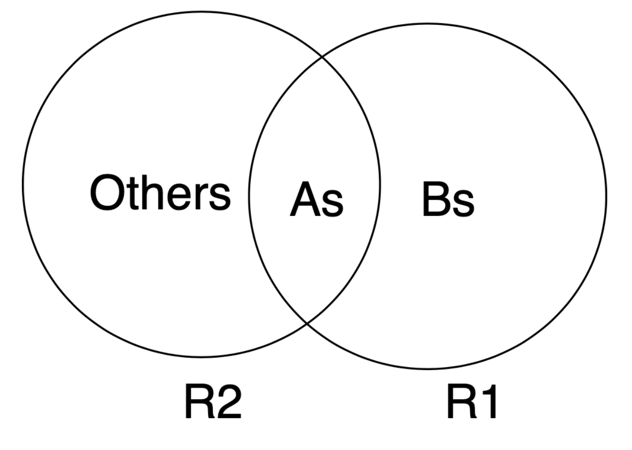
BCNF Decomposition

- Find a dependency that violates the BCNF condition: $X \rightarrow A$

$$\mathbf{A_1A_2...A_n \rightarrow B_1B_2....B_n}$$

- Heuristic : choose $B_1B_2...B_n$ "as large as possible", helps avoid unnecessarily fine-grained decomposition

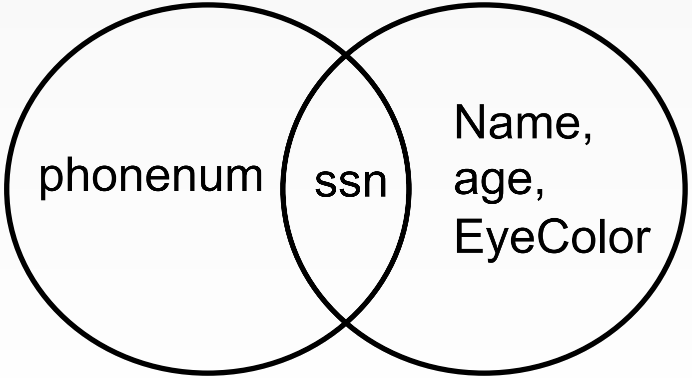
Decompose:



Continue until there are no BCNF violations left.

Example Decomposition

- Person (name, ssn, age, EyeColor, phonenum)
- FD: SSN → Name, age, EyeColor
- BCNF:
  - Person1 (ssn, phonenum)
  - Person2 (ssn, name, age, EyeColor)



Example Decomposition

- Courses(Number, DepartmentName, CourseName, Classroom, Enrollment, StudentName, Address)
- FD: Number, DeparmentName → CourseName, Classroom, Enrollment
- Compute {Number, DeparmentName}+ = {Number, DeparmentName, CourseName, Classroom, Enrollment}
  - R1 = {Number, DeparmentName, CourseName, Classroom, Enrollment}
  - R2 = {Number, DeparmentName, StudentName, Address}

Example Decomposition

- Students(ID, Name, AdvisorId, AdvisorName, FavouriteAdvisorId)
- FD:
  - ID → Name, FavouriteAdvisorId
  - AdvisorId → AdvisorName
- {ID, AdvisorId} is the key
- Compute ID+ = {ID, Name, FavouriteAdvisorId}
- Decompose R into
  - R1 = **{ID, Name, FavouriteAdvisorId}**
  - R2 = {ID, AdvisorId, AdvisorName}
- Decompose R2 into
  - R3 = **{AdvisorId, AdvisorName}**
  - R4 = **{ID, AdvisorId}**

Example Decomposition

- Person (Name, ssn, Age, EyeColor, phonenum, Draftworthy)
- F = {SSN → Name, Age, EyeColor, Age → Draftworthy }
- SSN → Name, Age, EyeColor,Draftworthy
  - Compute SSN+ = {SSN, Name, Age, EyeColor, Draftworthy}
  - Decompose R into
    - R1 = SSN+ = {SSN, Name, Age, EyeColor, Draftworthy}
    - R2 = SSN ∪ (R – SSN+) = {SSN} ∪ {phonenum} = **{SSN, phonenum}**
- Age → Draftworthy
  - Compute Age+ = {Age, Draftworthy}
  - Decompose R1 into
    - R3 = Age+ = **{Age, Draftworthy}**
    - R4 = Age ∪ (R1 – Age+) = {Age} ∪ {SSN, Name, EyeColor} = **{Age, SSN, Name, EyeColor}**

VIRGINIA TECH

Two-attribute relations

- Let A and B be the only two attributes of R
- Claim: R is in BCNF.
- If A → B is true, B → A is not:
  - A → B does not violate BCNF
- If B → A is true, A → B is not:
  - Symmetric
- If A → B is true, B → A is true:
  - Both are keys, therefore neither violate BCNF

Is BCNF Decomposition unique?

- R(SSN, netid, phone)
- FD1: SSN → netid
- FD2: netid → SSN
- If we do FD1 first:
  - (SSN, netid) and (SSN, phone)
- If we do FD2 first:
  - (netid, SSN) and (netid, phone)

Summary BCNF

- BCNF: each field contains data that cannot be inferred via FDs
  - ensuring BCNF is a good heuristic.
- Not in BCNF? Try decomposing into BCNF relations
- BCNF removes certain types of redundancies
  - for examples of redundancy that it cannot remove, see "multi-valued redundancy" (Addressed by 4NF, see textbook)
- BCNF decomposition **avoids information loss**
  - You can construct the original relation instance from the decomposed relations' instances
- Downside of BCNF: not all dependencies are preserved (some are split across relations)
  - If you want to preserve dependencies, you will have redundancy (Tradeoff!)

Lossless Decomposition but Lose dependencies

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

⇒

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

A → B; C → B

A	C
1	3
4	6
7	8

⋈

B	C
2	3
5	6
2	8

=

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

- But, now we can't check A → B without doing a join!

First Normal Form (1NF)

- All attributes are atomic (ie., no set-valued attr., a.k.a. ‘repeating groups’)
- Each attribute name must be unique.
- Each attribute value must be single.
- Each row must be unique.

1NF?

- All attributes are atomic (ie., no set-valued attr., a.k.a. ‘repeating groups’)
- Each attribute name must be unique.
- Each attribute value must be single.
- Each row must be unique.

Topic	Student	Grade
Intro to DBMS	Joe	A
	Sue	B
Java	Zhen	C
	Sally	D

1NF - No

- Each attribute name must be unique.
- Each attribute value must be single.
- Each row must be unique.

Topic	Student	Grade
Intro to DBMS	Joe	A
	Sue	B
Java	Zhen	C
	Sally	D

1NF

- Each attribute name must be unique.
- Each attribute value must be single.
- Each row must be unique.

Topic	Student	Grade
Intro to DBMS	Joe	A
Intro to DBMS	Sue	B
Java	Zhen	C
Java	Sally	D

Topic, Student -> Grade

Second Normal Form (2NF)

- Table is already in 1NF
- No non-key attribute is dependent on any proper subset of the key
  - All partial dependencies are moved to another table.

2NF

- 1NF + No non-key attribute is dependent on any proper subset of the key (i.e. no partial dependencies).

StudentID	ProjectID	StudentName	ProjectName
S89	P09	Olivia	Geo Location
S76	P07	Jacob	Cluster Exploration
S56	P03	Ava	IoT Devices
S92	P05	Alexandra	Cloud Deployment

StudentID and **ProjectID are key**

2NF

StudentID	ProjectID	StudentName
S89	P09	Olivia
S76	P07	Jacob
S56	P03	Ava
S92	P05	Alexandra

ProjectID	ProjectName
P09	Geo Location
P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

Third Normal Form (3NF)

- Table is already in 2NF.
- Nonprimary key attributes do not depend on other nonprimary key attributes
(i.e. no transitive dependencies)
  - All transitive dependencies are moved into another table.

3NF

- 2NF + No Transitive Dependencies

StudyID	Course Name	Teacher Name	Teacher Tel
1	Database	Sok Piseth	012 123 456
2	Database	Sao Kanha	0977 322 111
3	Web Prog	Chan Veasna	012 412 333
4	Web Prog	Chan Veasna	012 412 333
5	Networking	Pou Sambath	077 545 221

StudyId -> CourseName, TeacherName, TeacherTel

TeacherName -> TeacherTel

StudyID	Course Name	Teacher Name	Teacher Tel
1	Database	Sok Piseth	012 123 456
2	Database	Sao Kanha	0977 322 111
3	Web Prog	Chan Veasna	012 412 333
4	Web Prog	Chan Veasna	012 412 333
5	Networking	Pou Sambath	077 545 221

StudyId -> CourseName, TeacherName, TeacherTel

TeacherName -> TeacherTel

StudyID	Course Name	Teacher Name
1	Database	Sok Piseth
2	Database	Sao Kanha
3	Web Prog	Chan Veasna
4	Web Prog	Chan Veasna
5	Networking	Pou Sambath

Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Pou Sambath	077 545 221

3NF – Your turn

Library_Patron(Id, Name, Fines, BookId, BookName)

Id	Name	Fines	BookId	BookName
1	Joe	0.00	ABCD	The Cat in the Hat
2	Sally	1.00	DEFG	Fox in Socks

Id -> Name Fines BookId
BookId -> BookName

3NF – Your turn

Library_Patron(Id, Name, Fines, BookId)

Id	Name	Fines	BookId
1	Joe	0.00	ABCD
2	Sally	1.00	DEFG

Books(BookId, BookName)

BookId	BookName
ABCD	The Cat in the Hat
DEFG	Fox in Socks

Third Normal Form (3NF)

- Recall BCNF: not all dependencies are preserved
- Reln R with FDs F is in 3NF if, for all $X \rightarrow A$ in F+
  - $A \in X$ (called a trivial FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is prime")

- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some **redundancy** is possible.
- It is a compromise, **used when BCNF not achievable**
  - (e.g., no "good" decomp, or performance considerations).
  - Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.

- **To ensure dependency preservation, one idea:**
  - **If $X \rightarrow Y$ is not preserved, add relation XY.**

  Problem is that XY may violate 3NF!

  e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?

- **Refinement**: Instead of the given set of FDs F, use a ***minimal cover for F***.

Recall: Minimal Cover for a Set of FDs

- ***Minimal cover*** G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and '***as small as possible'*** in order to get the same closure as F
- e.g., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$

Normal forms - 3NF

how to bring a schema to 3NF?

two algo’s in book: First one:

- start from ER diagram and turn to tables
- then we have a set of tables R1, ... Rn which are in 3NF
- for each FD (X->A) in the cover that is not preserved, create a table (X,A)

Normal forms - 3NF

how to bring a schema to 3NF?

two algo's in book: Second one ('synthesis')

- take all attributes of R
- for each FD (X->A) in the cover, add a table (X,A)
- if not lossless, add a table with appropriate key

We prefer Synthesis as it is clearer and does not need ER diagrams

3NF Synthesis Algorithm

- Let F be the set of all FDs of R.
- We will compute a lossless-join, dependency-preserving decomposition of R into S, where every relation in S is in 3NF.

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD $X \rightarrow A$ in G, use $X \cup A$ as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

3NF Synthesis Algorithm

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD $X \rightarrow A$ in G, use $X \cup A$ as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

Students (FirstName, Hall, Address)

F = { FirstName->Hall, Address; Hall -> Address }

VIRGINIA TECH™

3NF Synthesis Algorithm

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD $X \rightarrow A$ in G, use $X \cup A$ as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

Students (FirstName, Hall, Address)
F = { FirstName->Hall, Address; Hall -> Address }

1. Minimal Basis Fc: { FirstName -> Hall, Hall -> Address }

VIRGINIA TECH

3NF Synthesis Algorithm

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD X → A in G, use X ∪ A as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

Students (FirstName, Hall, Address)
F = { FirstName->Hall, Address; Hall -> Address }

1. Minimal Basis: { FirstName -> Hall, Hall -> Address }
2. Keys for R: {FirstName}

3NF Synthesis Algorithm

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD $X \rightarrow A$ in G, use $X \cup A$ as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

Students (FirstName, Hall, Address)
F = { FirstName->Hall, Address; Hall -> Address }

1. Minimal Basis: { FirstName -> Hall, Hall -> Address }
2. Keys for R: {FirstName}
3. New Relations: Names(FirstName, Hall), Halls(Hall, Address)

3NF Synthesis Algorithm

1. Find a minimal basis (canonical cover) for F, say G.
2. Find all keys for R.
3. For every FD X → A in G, use X ∪ A as the schema for one of the relations in S.
4. If the attributes in none of the relations in S form a superkey for R, add another relation to S whose schema is a key for R.

Students (FirstName, Hall, Address)
F = { FirstName->Hall, Address; Hall -> Address }

1. Minimal Basis: { FirstName -> Hall, Hall -> Address }
2. Keys for R: {FirstName}
3. New Relations: Names(FirstName, Hall), Halls(Hall, Address)
4. Are the attributes of Names or Halls a superkey for Students?

Example: 3NF

Example:

R: ABC

F: A->B, C->B

- Q1: what is the cover?

- Q2: what is the decomposition to 3NF?

Example: 3NF

Example:

R: ABC

F: A->B, C->B

- Q1: what is the cover?

A1: ‘F’ is the cover

- Q2: what is the decomposition to 3NF?

Example: 3NF: Step 1

Example:

R: ABC

F: A->B, C->B

- Q1: what is the cover?
  - A1: ‘F’ is the cover
- Q2: what is the decomposition to 3NF?
  - A2: one table each for the FDs
  - R1(A,B), R2(C,B), ...
  - But is it lossless?? Or equivalently do any of the relations
  - in S form a superkey for R?

Example: 3NF: Step 2

Example:

R: ABC

F: A->B, C->B

- Q1: what is the cover?
  - A1: ‘F’ is the cover
- Q2: what is the decomposition to 3NF?
  - A2: R1(A,B), R2(C,B), R3(A,C)
  - (note that AC is a key for R)

Normal forms - 3NF vs BCNF

- If ‘R’ is in BCNF, it is always in 3NF (but not the reverse)
- In practice, aim for
  - BCNF; lossless join; and dep. preservation
- if impossible, we accept
  - 3NF; but insist on lossless join and dep. preservation

Why Normalization ?

- By limiting redundancy, normalization helps maintain consistency and saves space.
- But performance of querying can suffer because related information that was stored in a single relation is now distributed among several.
- *Sometimes* you will de-normalize for the sake of performance.
  - But do so cautiously and intelligently.

Normal Forms

Form	Requirement
1NF	Each attribute name must be unique. Each attribute value must be single. Each row must be unique.
2NF	1NF no non-key attribute is dependent on any proper subset of the key
3NF	2NF No transitive dependencies
BCNF	3NF All determinants are superkeys
4NF	BCNF No multi-valued dependencies
5NF, 6NF	Who cares.. ☺

Summary

- What to do when a lossless-join, dependency preserving decomposition into BCNF is impossible?
  - There is a more *permissive* Third Normal Form (3NF)
  - But you'll have redundancy. Beware. You will need to keep it from being a problem in your application code.

- Note: even more *restrictive* Normal Forms exist
  - we don't cover them in this course, but some are in the book

Reading and Next Class

- BCNF, 3NF and Normalization: Ch 19.4-19.9
- Next: ACID and Transactions: Ch 16.1 – 16.6